

NASA-CR-194589

# 3rd NASA Symposium on VLSI Design

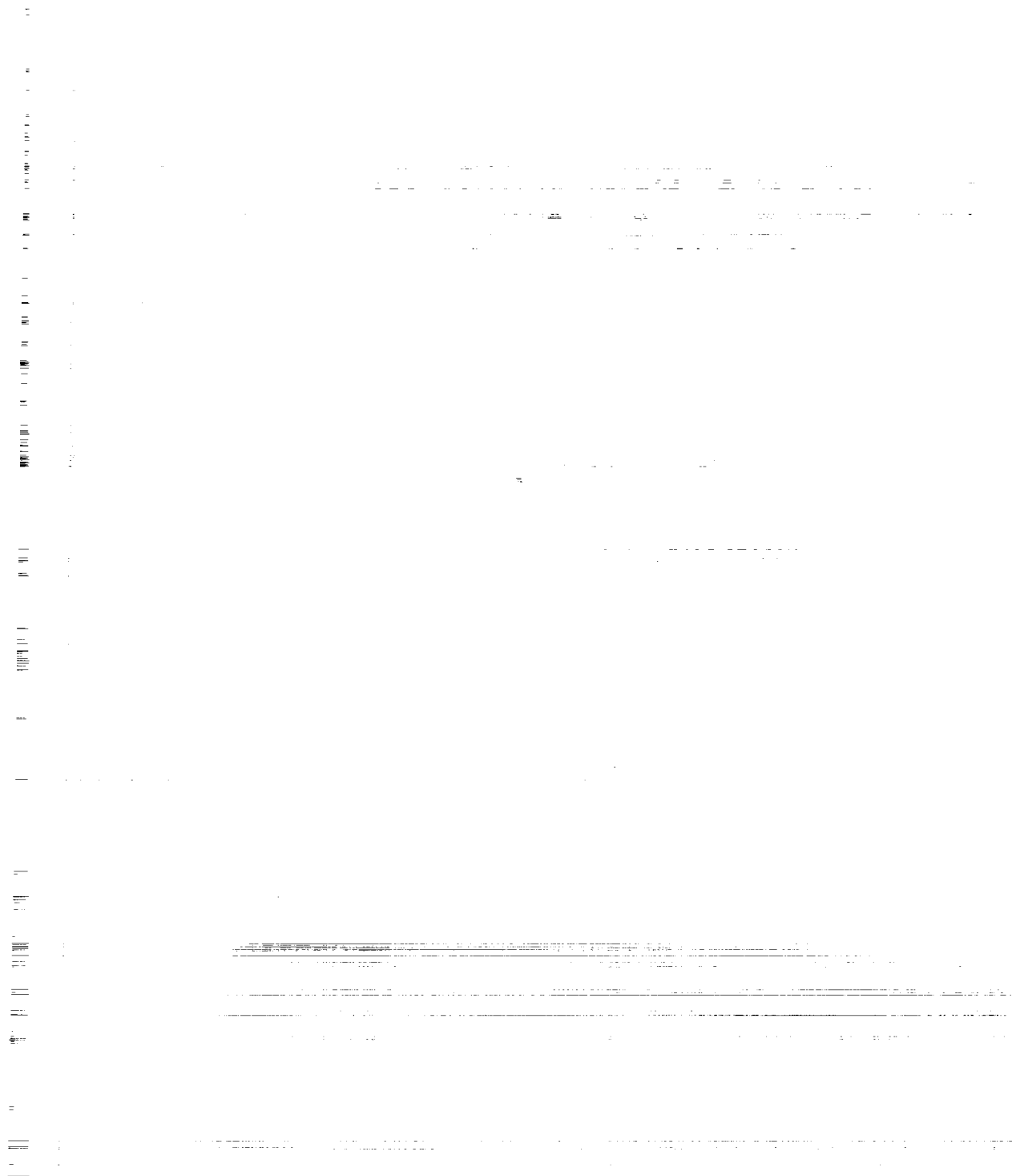
University of Idaho  
Moscow, Idaho

October 30-31, 1991

(NASA-CR-194589) THE 1991 3RD NASA  
SYMPOSIUM ON VLSI DESIGN (Idaho  
Univ.) 492 p

N94-18337  
--THRU--  
N94-18381  
Unclas

G3/33 0191397



Cover design by Peter Vincent



Welcome to the third annual NASA Symposium on VLSI Design, co-sponsored by the IEEE. Each year this symposium is organized by the NASA Space Engineering Research Center (SERC) at the University of Idaho and is held in conjunction with a quarterly meeting of the NASA Data System Technology Working Group (DSTWG). One task of the DSTWG is to develop new electronic technologies that will meet next generation electronic data system needs. The symposium provides insights into developments in VLSI and digital systems which can be used to increase data systems performance.

The NASA SERC is proud to offer, at its third symposium on VLSI design, presentations by an outstanding set of individuals from national laboratories, the electronics industry and universities. These speakers share insights into next generation advances that will serve as a basis for future VLSI design.

Interest in the conference has increased with 46 papers in 8 categories included in this years proceedings. National Laboratories are represented by Lawrence Livermore Laboratory and the Johns Hopkins University Applied Physics Laboratory. Private industry is represented by Hewlett Packard-CTG, Hewlett Packard-ICBD, Advanced Hardware Architectures, Smith International Inc., and United Technologies Microelectronics Center. Universities are represented by Brigham Young University, Montana State University, Washington State University, University of Calgary, University of Western Australia, University of Houston, Stanford University, Ecole Polytechnique de Montreal, Concordia University, University of California at Davis, University of British Columbia, Portland State University, University of Madras, Old Dominion University and the University of Idaho. In addition we are happy to welcome a number of papers presented by international authors.

There are individuals whose assistance was critical to the success of this symposium. Barbara Martin worked long hours to assemble the conference proceedings. Judy Wood did another excellent job at coordinating the many conference activities. Sterling Whitaker organized the symposium. The efforts of these professionals were vital and are greatly appreciated.

I am encouraged by the growth we have experienced in this years symposium and look for suggestions that will allow a better symposium next year. I hope you enjoy your stay in Moscow, Idaho and I extend an invitation to visit MRC research laboratories during the symposium.

Gary K. Maki

## **Session 1 – Featured Presentations I**

**Chairman: Gary Maki**

- |  |            |
|--|------------|
| <b>Experience with Custom Processors in Space Flight Applications</b>            | <b>1.1</b> |
| M. Fraeman, J. Hayes, D. Lohr, B. Ballard, R. Williams and R. Henshaw            |            |
| <b>Multi-chip Modules: A High-Performance Packaging Alternative</b>              | <b>1.2</b> |
| L. Salmon  |            |
| <b>New Dynamic FET Logic and Serial Memory Circuits for VLSI GaAs Technology</b> | <b>1.3</b> |
| A. Eldin   |            |

## **Session 2 – VLSI Circuit Design**

**Chairman: Kelly Cameron**

- |   |            |
|---|------------|
| <b>Automated ILA Design for Synchronous Sequential Circuits</b>                   | <b>2.1</b> |
| M. Liu, K. Liu, G. Maki and S. Whitaker   |            |
| <b>Single Phase Dynamic CMOS PLA Using Charge Sharing Technique</b>               | <b>2.2</b> |
| Y. Dhong and C. Tsang   |            |
| <b>An SEU Immune Logic Family</b>   | <b>2.3</b> |
| J. Canaris  |            |
| <b>Cellular Logic Array for Computation of Squares</b>                            | <b>2.4</b> |
| M. Shamanna, S. Whitaker and J. Canaris   |            |
| <b>Fault Tolerant Sequential Circuits Using Sequence Invariant State Machines</b> | <b>2.5</b> |
| M. Alahmad and S. Whitaker  |            |



## **Session 3 – VLSI Architectures I**

**Chairman: Jon Gibson**

<b>VLSI Architectures for Geometrical Mapping Problems in High Definition Image Processing</b> K. Kim and J. Lee	3.1
<b>Performance of Defect Tolerant Set Associative Cache Memories</b> J. Frenzel	3.2
<b>SPROC - A Multiple-Processor DSP IC</b> R. Davis	3.3
<b>An Extended Reed Soloman Decoder Design</b> J. Chen, P. Owsley and J. Purviance	3.4
<b>An Improved Distributed Arithmetic Architecture</b> X. Guo and D. Lynn	3.5

## **Session 4 – Featured Presentations II**

**Chairman: Sterling Whitaker**

<b>An Analog Retina Model for Detecting Dim Moving Objects Against a Bright Moving Background</b> R. Searfus, M. Colvin, F. Eeckman and T. Axelrod	4.1
<b>Low Power Signal Processing Research at Stanford</b> J. Burr, P. Williamson and A. Peterson	4.2
<b>Technology Design, Simulation and Evaluation for SEP-Hardened Circuits</b> J. Adams, D. Allred, M. Barry, J. Silver, P. Rudeck and C. Hafer	4.3

## **Session 5 – Neural Networks**

**Chairman: John Purviance**

<b>Pulse-Firing Winner-Take-All Networks</b>	<b>5.1</b>
J. Meador	
<b>Training Product Unit Neural Networks with Genetic Algorithms</b>	<b>5.2</b>
D. Janson, J. Frenzel and D. Thelen	
<b>VLSI Synthesis of Digital Application Specific Neural Networks</b>	<b>5.3</b>
G. Beagles and K. Winters	
<b>Measurement Selection for Parametric IC Fault Diagnosis</b>	<b>5.4</b>
A. Wu and J. Meador	

## **Session 6 – VLSI Architectures II**

**Chairman: Don Thelen**

<b>Fuzzy Control of Magnetic Bearings</b>	<b>6.1</b>
J. Feeley, G. Niederaurer and D. Ahlstrom	
<b>Direct Kinematics Solution Architectures for Industrial Robot Manipulators: Bit Serial Versus Parallel</b>	<b>6.2</b>
J. Lee and K. Kim	
<b>Simplified Microprocessor Design for VLSI Control Applications</b>	<b>6.3</b>
K. Cameron	
<b>A Modified Reconfigurable Data Path Processor</b>	<b>6.4</b>
G. Ganesh, S. Whitaker and G. Maki	

## **Session 7 – Featured Presentations III**

**Chairman: Joseph Feeley**

- |   |     |
|---|-----|
| <b>Systolic IC Array for Genetic Computation</b>                                | 7.1 |
| D. Anderson   |     |
| <b>High Performance Multiprocessor Architecture for a 3-D Lattice Gas Model</b> | 7.2 |
| F. Lee, M. Flynn and M. Morf  |     |
| <b>Verification of VLSI Designs</b>   | 7.3 |
| P. Windley  |     |

## **Session 8 – Verification I**

**Chairman: Phil Windley**

- |   |     |
|---|-----|
| <b>A New Variable Testability Measure</b>   | 8.1 |
| M. Jamoussi, B. Kaminska and D. Mulkhedkar  |     |
| <b>Controlling State Explosion During Automatic Verification of Delay-Insensitive and Delay Constrained VLSI Systems Using the POM Verifier</b> | 8.2 |
| D. Probst and L. Jensen   |     |
| <b>Formal Verification of an MMU and MMU Cache</b>  | 8.3 |
| E. Schubert   |     |
| <b>Formal Hardware Verification of Digital Circuits</b>   | 8.4 |
| J. Joyce and C. Seger   |     |

## Session 9 – Analog Design

Chairman: Earl Gray

- High Accuracy Switched-Current Circuits Using An Improved Dynamic Mirror** 9.1  
G. Zweigle and T. Fiez
- A Tunable CMOS Constant Current Source** 9.2  
D. Thelen
- DC and Small-Signal Physical Models for the AlGaAs/GaAs High Electron Mobility Transistor** 9.3  
J. Sarker and J. Purviance

## Session 10 – Verification II

Chairman: Phil Windley

- Formal Specification of a High Speed CMOS Correlator** 10.1  
P. Windley
- A Verification Logic Representation of Indeterministic Signal States** 10.2  
J. Gambles and P. Windley
- Formal Verification of State Machines** 10.3  
M. Alahmad and P. Windley

## Session 11 – Design Innovations I

Chairman: Bill Smith

- Ultra Low Power CMOS Technology** 11.1  
J. Burr and A. Peterson
- Parallel Optimization Algorithms and Their Implementations in VLSI Design** 11.2  
G. Lee and J. Feeley
- Canonical Multivalued Input Reed-Muller Trees and Forms** 11.3  
M. Perkowski

## **Session 12 – Asynchronous Design**

**Chairman: Gary Maki**

- |  |      |
|--|------|
| <b>Asynchronous Sequential Circuit Design Using Pass Transistor Iterative Logic Arrays</b> | 12.1 |
| M. Liu, G. Maki and S. Whitaker  |      |
| <b>Pulse Mode VLSI Asynchronous Circuits</b>   | 12.2 |
| Q. Chen and G. Maki  |      |
| <b>Improved Self-Arbitrated VLSI Asynchronous Circuits</b>                                 | 12.3 |
| P. Winterrowd  |      |

## **Session 13 – Design Innovations II**

**Chairman: Kel Winters**

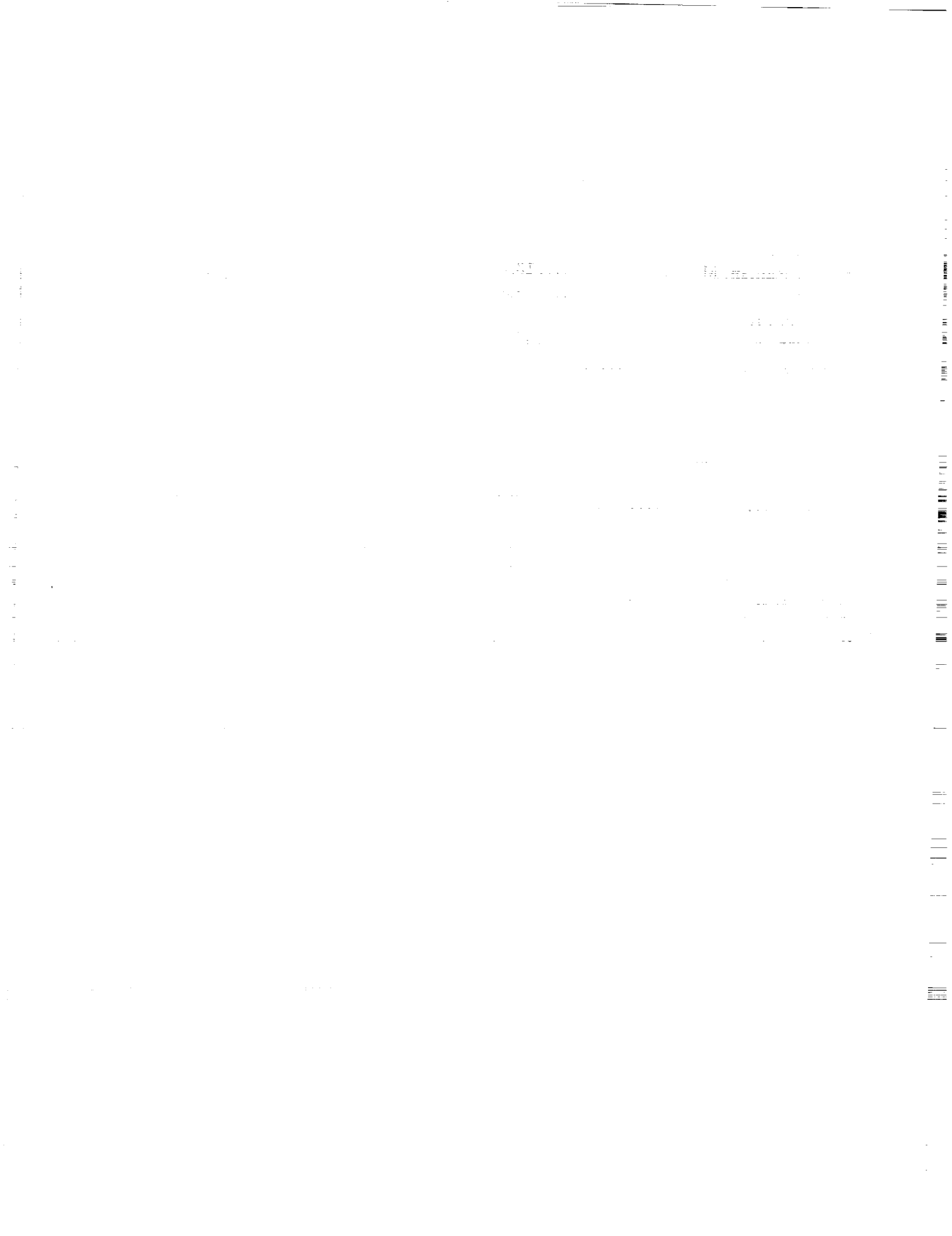
- |  |      |
|--|------|
| <b>A Special Purpose Silicon Compiler for Designing Supercomputing VLSI Systems</b>  | 13.1 |
| P. Murugavel, V. Kamakoti, M. Shankar Raman, S. Rangarajan, M. Mallikarjun, B. Karthikeyan, T. Prabhakar, V. Satish, P. Venkatasubramaniam, R. Sivakumar, R. Srinivasan, S. Chandrasekhar, G. Suresh, M. Karthikeyan, S. Ramachandran, S. Sankar, P. Balaji and P. Kishore |      |
| <b>A Fast Adaptive Convex Hull Algorithm on Two-Dimensional Processor Arrays with a Reconfigurable Bus System</b>  | 13.2 |
| S. Olariu, J. Schwing and J. Zhang   |      |
| <b>VHDL Simulation with Access to Transistor Models</b>  | 13.3 |
| J. Gibson  |      |

## List of Authors

Adams, J. ....	4.3	Kamakoti, V. ....	13.1
Ahlstrom, D. ....	6.1	Kaminska, B. ....	8.1
Alahmad, M. ....	2.4	Karthikeyan, B. ....	13.1
.....	10.3	Karthikeyan, M. ....	13.1
Allred, D. ....	4.3	Kim, K. ....	3.1
Anderson, D. ....	7.1	.....	6.2
Axelrod, T. ....	4.1	Kishore, P. ....	13.1
Balaji, P. ....	13.1	Lee, F. ....	7.2
Ballard, B. ....	1.1	Lee, G. ....	11.2
Barry, M. ....	4.3	Lee, J. ....	3.1
Beagles, G. ....	5.3	Lee, J. ....	6.2
Burr, J. ....	4.2	Liu, K. ....	2.1
Burr, J. ....	11.1	Liu, M. ....	2.1
Cameron, K. ....	6.3	.....	12.1
Canaris, J. ....	2.3	Lohr, D. ....	1.1
.....	2.4	Lynn, D. ....	3.5
Chandrasekhar, S. ....	13.1	Maki, G. ....	2.1
Chen, J. ....	3.4	.....	6.4
Chen, Q. ....	12.2	.....	12.1
Colvin, M. ....	4.1	.....	12.2
Davis, R. ....	3.3	Mallikarjun, M. ....	13.1
Dhong, Y. ....	2.2	Meador, J. ....	5.1
Eeckman, F. ....	4.1	.....	5.4
Eldin, A. ....	1.3	Morf, M. ....	7.2
Feeley, J. ....	6.1	Mulkhedkar, D. ....	8.1
.....	11.2	Murugavel, P. ....	13.1
Fiez, T. ....	9.1	Niederaurer, G. ....	6.1
Flynn, M. ....	7.2	Olariu, S. ....	13.2
Fraeman, M. ....	1.1	Owsley, P. ....	3.4
Frenzel, J. ....	3.2	Perkowski, M. ....	11.3
.....	5.2	Peterson, A. ....	4.2
Gambles, J. ....	10.2	.....	11.1
Ganesh, G. ....	6.4	Prabhakar, T. ....	13.1
Gibson, J. ....	13.3	Probst, D. ....	8.2
Guo, X.. ....	3.5	Purviance, J. ....	3.4
Hafer, C. ....	4.3	.....	9.3
Hayes, J. ....	1.1	Ramachandran, S. ....	13.1
Henshaw, R.. ....	1.1	Rangarajan, S. ....	13.1
Jamoussi, M. ....	8.1	Rudeck, P. ....	4.3
Janson, D. ....	5.2	Salmon, L. ....	1.2
Jensen, L. ....	8.2	Sankar, S. ....	13.1

## 3rd NASA Symposium on VLSI Design

Sarkar, J. ....	9.3
Satish, V. ....	13.1
Schubert, E. ....	8.3
Schwing, J. ....	13.2
Searfus, R. ....	4.1
Shamanna, M. ....	2.4
Shankar Raman, M. ....	13.1
Silver, J. ....	4.3
Sivakumar, R. ....	13.1
Srinivasan, R. ....	13.1
Suresh, G. ....	13.1
Thelen, D. ....	5.2
.....	9.2
Tsang, C. ....	2.2
Venkatasubramaniam, P. ....	13.1
Whitaker, S. ....	2.1
.....	2.4
.....	2.5
.....	6.4
.....	12.1
Williams, R. ....	1.1
Williamson, P. ....	4.2
Windley, P. ....	7.3
.....	10.1
.....	10.2
.....	10.3
Winterrowd, P. ....	12.3
Winters, K. ....	5.3
Wu, A. ....	5.4
Zhang, J. ....	13.2
Zweigle, G. ....	9.1





## Experience with Custom Processors in Space Flight Applications

M. E. Fraeman, J. R. Hayes, D. A. Lohr, B. W. Ballard,  
R. L. Williams, and R. M. Henshaw  
Johns Hopkins University Applied Physics Laboratory  
Laurel, Maryland 20723

*Abstract-* APL has developed a magnetometer instrument for a Swedish satellite named Freja with launch scheduled for August 1992 on a Chinese Long March rocket. The magnetometer controller utilized a custom microprocessor designed at APL with the Genesil silicon compiler. The processor evolved from our experience with an older bit-slice design and two prior single chip efforts. The architecture of our microprocessor greatly lowered software development costs because it was optimized to provide an interactive and extensible programming environment hosted by the target hardware. Radiation tolerance of the microprocessor was also tested and was adequate for Freja's mission—20 kRad(Si) total dose and very infrequent latch-up and single event upset events.

### 1 Introduction

The Johns Hopkins University Applied Physics Laboratory (APL) has developed a microprocessor that is well suited to one-of-a-kind embedded applications especially in satellite instrument control. The chip has been qualified for use in a magnetometer instrument for the Swedish Freja satellite. The processor's language directed architecture reduced Freja software costs because the flight hardware served as its own development system. Thus, unlike traditional interpreted programming languages like Basic, Lisp, or Smalltalk, our Forth language development system was fully supported on the embedded flight processor. Performance was also equivalent or better than that obtained by other microprocessors programmed in languages like C with traditional cross-compilers and development systems.

Our experiences using Forth to program spacecraft instrumentation computers, and our early efforts to design a 32-bit microprocessor specifically intended to execute Forth code are described in this paper. The design, architecture, and performance of our most recent version of this microprocessor, called the SC32<sup>1</sup>, are summarized in Section 4. Discussion of our use of the SC32 in the Freja magnetometer includes our efforts to qualify the microprocessor for space flight. Finally, we discuss some of the lessons we learned using a custom designed integrated circuit in space flight hardware.

---

<sup>1</sup>The SC32 has been commercially licensed by Silicon Composers, Inc., Palo Alto, Ca. They offer chips, board level development systems, and support software.

Table 1: APL Forth-based Subsystems and Experiments

SPACECRAFT	SUBSYSTEM/EXPERIMENT	LAUNCH DATE	PROCESSOR
MAGSAT	Attitude Control	6/79	RCA 1802
DMSP	Magnetometer	(classified)	RCA 1802
HILAT	Magnetometer	6/83	RCA 1802
Polar Bear	Magnetometer	6/83	RCA 1802
Astro-1	Ultraviolet Telescope(HUT)	12/90	AMD 2900
Freja	Magnetometer	8/92 (est)	SC32

## 2 Background

### 2.1 Forth

Forth has an extremely simple syntax so only a trivial parser is needed to allow it to run in impoverished hardware environments. Lexical properties are also simple. Forth subroutines, called words, are delimited by spaces. The words themselves can consist of any characters other than the delimiter. This simplicity keeps the interpreter small allowing full featured Forth systems to fit comfortably in as little as 8 kbytes of memory.

Programming in Forth consists of defining new words in terms of existing words. The new word is incrementally compiled and can be invoked interactively by the programmer. Thus, the usual benefits of interpreted languages are reaped, especially simplified testing and a resulting higher confidence in program correctness.

### 2.2 APL Space Applications of Forth

Table 1 summarizes APL's experience with spacecraft instrumentation we have developed and programmed using Forth.[1] We have also used the language on other projects including ground support equipment and control of laboratory instrumentation. Application tasks ranged from relatively simple data acquisition functions to control of the complex, space shuttle based Hopkins Ultraviolet Telescope (HUT)—one of three ultraviolet telescopes (all programmed in Forth) that comprised the Astro-1 mission at the end of 1990. Our most recent instrument, a magnetometer for the Swedish Freja satellite will be described later in this paper.

Our earliest space flight applications were based on the relatively simple RCA 1802 microprocessor. But during the early definition of the HUT command and data handling system around 1980, it became clear that a far more powerful processor was needed to satisfy that project's requirements. After exploring an architecture based on as many as four TI 9900 microprocessors (the fastest microprocessor qualified for space that was then available), we realized that a single faster machine would have numerous advantages. The software would be easier to write and test, and more importantly, uniprocessor code and hardware would be more flexible in the face of evolving requirements and as system

interfaces were more clearly defined.

## 2.3 The Hopkins Ultraviolet Telescope Processor

The AMD 2900 bit-slice component family was used to build a 16-bit computer that implemented Forth's primitive operations directly in microcode. In the early 1980s, this was the only way we could build a single processor with throughput that met our requirements and that also could be qualified for use in space. Our bit-slice processor was able to compile and execute Forth interactively, even on the flight unit, without needing extensive support tools. Performance was also very good (approximately 500,000 Forth operations per second) which allowed us to design an unusually flexible software system. The final flight software required about 5 person-years of development time (including developing the detailed software requirements), contained 29 cooperating concurrent processes, and consisted of about 12,000 lines of Forth code and comments.

We gained valuable experience with Forth based computers while developing, using, and flying the HUT processor. A fast computer that supported a compact but interactive and extensible software development system on flight hardware had many advantages. It encouraged the development of powerful yet flexible software while minimizing the costs of writing, testing, and maintaining that code. However, HUT also showed that the 64 Kword address space of 16-bit machines was inadequate for larger embedded systems. Towards the end of the development cycle flight processor memory became too full to support an interactive environment so we had to fall back on clumsier traditional cross-compiler based methodology.

## 3 The FRISC Project

At the same time our work on HUT hardware was winding down in 1984, we were also initiating an effort to develop experience in VLSI design. We combined our experience in Forth computers and our interest in VLSI into an effort to develop a 32-bit Forth microprocessor. During 1985 we developed the processor architecture that we called FRISC (Forth Reduced Instruction Set Computer) and ported VLSI design tools developed at several universities a 68010 based workstation.

### 3.1 FRISC 1

By the beginning of 1986, with tools and architecture firmly in hand, we started detailed design of a chip that implemented most of our ideas. This was FRISC 1, the first in a series of chips that evolved into the SC32. We targeted the 4  $\mu\text{m}$  Silicon on Sapphire (SOS) process then available through MOSIS. We selected SOS technology for several reasons. First, SOS is inherently immune to radiation induced latch-up and would thus be a candidate technology for future integrated circuits used in flight systems. The absence of active-substrate junction capacitance reduces load and hence improves speed. Circuit density is improved because there is no minimum p-active—n-active separation design rule.

#### 1.1.4

Finally, on a more practical note, the SOS process was available through MOSIS at no cost as far as the project's budget was concerned. So the chance to get experience with a technology with significant benefits for chips intended for use in space was too good to pass up.

Design of the 18,000 transistor chip was completed by mid-April 1986. It easily fit inside a standard MOSIS 7.9 mm x 9.2 mm pad frame. We used *caesar* for layout, *lyra* for design rule checking, *rnl* for functional simulation, *spice* for circuit simulation, and the usual collection of customized shell scripts, format translators, and system utilities for coordinating the design team's work. While the chips were being fabricated, we built a wire wrapped Multibus CPU board with memory and a programmable non-overlapping clock generator board to test our parts.

Three months later we had our chips and began to test them. About half of the parts that were eventually delivered appeared to function except that one data bit was always stuck high. Unfortunately, that specific bit was used in the instruction set to cause the processor to output a value, so we had no way to inspect the contents of the chip's registers. Microscopic analysis later revealed a spacing design rule violation at the interface between the pad ring cell and the cell containing the chip's interior logic. This error was undetected because *lyra* flattened the layout of intersecting areas on adjacent cells after checking the cells individually. Our design hierarchy consisted of the pad ring in one cell and all the other circuitry in a second cell completely enclosed by the ring. Therefore the top level rule check flattened the entire design and greatly exceeded the maximum virtual memory space supported by our host workstation so our mistake went undetected.

Despite this layout error, one chip was fully functional and we were able to demonstrate a full Forth system running on our own custom 32-bit microprocessor. But before we could submit a corrected design, MOSIS announced that they would no longer offer access to SOS.

## 3.2 FRISC 2

At the beginning of 1987, we started to redesign our chip with the MOSIS scalable ( $3\ \mu\text{m}$  to  $1.2\ \mu\text{m}$ ) bulk CMOS process. We also used the *magic* layout editor instead of *caesar* but still depended on *rnl* for switch level simulation. By April we sent the layout to MOSIS for a 20,000 transistor chip that implemented almost all of our original architecture. The active area for this chip, designed with  $3\ \mu\text{m}$  feature sizes, was slightly smaller than the previous version but it still required a 7.9 mm x 9.2 mm pad frame. However, an inadvertently grounded substrate prevented that part from working. Using a combination of infrared microphotography and careful inspection of the layout in the hot region we eventually located the error.<sup>2</sup> Since we made our mistake, a circuit extractor called *meztra*, was modified at the University of Washington to specifically detect similar errors. Apparently we weren't the first, and based on errors we've detected in other designs, not the last group to make a substrate connection error.

<sup>2</sup>This error has since been missed by dozens of students taking the midterm exam in a JHU VLSI design class.

A corrected layout was fabricated shortly thereafter and was fully functional. The fixed FRISC 2 could execute about 2.5 million Forth primitives per second (about five times faster than 25 MHz Motorola MC68020 running Forth) and consumed 150 mW. However, this performance was about twice as slow as we expected due to an incorrectly sized control line driver.

## 4 The SC32

While our efforts had eventually produced a functional and usable microprocessor, we did not reach our design goals on first silicon. In fact, we felt that our small team would not be able to build chips much more complex than FRISC 2 with the tools and workstations we used for that design. Furthermore, full logical and parametric functionality would probably be achieved only after several fabrication iterations. Our simulations were not as thorough as we would have liked since our workstation required a day to complete a switch level simulation of the execution of a few machine instructions. Determining the impact of more than one or two architectural alternatives on chip speed and area was impractical. Irregular structures such as control logic were very tedious to layout. Minor changes in control logic would often result in days of work to resimulate and update the layout. As our speed problem with FRISC 2 demonstrated, these structures were also a likely source of parametric as well as functional errors.

### 4.1 Genesil

Rather than waiting several years for workstation speeds to improve before tackling more complex chip designs, we investigated commercial VLSI design tools. Silicon Compilers Inc. (now part of Mentor Graphics, Inc.) had just released the Genesil silicon compiler. This was a fully integrated set of VLSI tools that let the user describe, implement, and analyze a design at the block diagram level.

Genesil's intended market was logic designers with no VLSI experience. Yet we were attracted to it because the compiler allowed a user to easily and quickly investigate the implications of many architectural alternatives. We felt that the greatest improvements in system performance could be gained by optimizing architecture while lower level enhancements would be of secondary importance. Any inefficiencies introduced by the high level design tool should be more than compensated for by the better architecture that the silicon compiler would allow the designer to develop. Genesil also automated many of the most time consuming aspects of VLSI design so a small team would be able to tackle larger projects. Thus we hoped that Genesil would be the better tool that would let our small team tackle larger designs.

### 4.2 SC32 Design

Genesil was installed at our site by June 1987, and we started using it to explore approaches to implementing our Forth architecture. We also enhanced our computer's architecture

based on the experience we gained on our earlier designs. The greater complexity that Genesil let us tackle with the same size team (2-3 part time people) also allowed us to improve the architecture. By mid-November we had completed our Genesil design work including thorough simulations of thousands of instructions. But due to delays in design verification at Silicon Compilers, our mask level design wasn't delivered to the foundry until February, 1988. After an extra month delay caused by problems with test vector formats, we received fully functional tested parts in May. The next day we had a single board computer running an interactive Forth development system.

We consider this third version of our Forth processor a complete success. It was fabricated with a 2  $\mu$ m epitaxial CMOS n-well process, contained 35,000 transistors, and consumed 660 mW. The die was 9.9 mm x 9.6 mm and was packaged in an 84 pin ceramic pin grid array. Despite obvious inefficiencies in the overall chip layout, the processor still ran at 10 MHz. Because the processor architecture is optimized for Forth the comparatively slow clock rate speed still executed 8-12 million primitives per second—a throughput still unmatched by any other 32-bit microprocessor implementation of the language of which we are aware.

### 4.3 Architecture

The detailed architecture of the SC32 has been described elsewhere.[2] Briefly, the machine has a 32 bit word address architecture and an instruction set that can implement most Forth primitives in a single instruction. Flow control instructions specify an absolute destination address and execute in a single cycle with no delay slots. The machine's register set is organized into two top-of-stack caches with single cycle access within the instruction set to the top four locations of each stack. These on-chip caches support stack depths limited only by main memory with overflow and underflow events handled entirely by hardware. Less than 1% overhead is added to typical Forth programs by our approach to stack management. There are up to eight other utility and special purpose registers allowed. The data path allows arithmetic operations between these registers to be completed in a single cycle. A flexible load/store instruction format transfers data between registers and memory and can also be used to form literal values.

### 4.4 Performance

Measuring and comparing processor performance is always controversial—especially for a new architecture not supported by commonly used languages. Different implementations of Forth are also difficult to compare since there are no commonly used benchmark programs written in that language. Finally, it is only natural to ask how a Forth version of a program compares to an equivalent implementation in a more widely used language.

Since Forth is the only high level language available for the SC32, we took the approach of manually translating a set of small integer benchmark programs from C to Forth. These programs were collected by the Computer Systems Laboratory at Stanford University and have since been translated from their original Pascal into C. They have been widely used

to evaluate the performance of many computer systems.

Because the Stanford programs are small, they are generally considered "toy" benchmarks that provide overly optimistic results in comparison to similar tests made with larger codes. But several factors suggest that the translated benchmark suite will provide a conservative estimate of performance running large Forth programs.

Merely translating these programs into Forth produced very poor and uncharacteristic Forth code. Word definitions were extremely long and difficult to debug. This meant that the SC32's efficient call/return mechanism was not used. However, our measurements showed that real Forth programs greatly benefit from this feature of the SC32. Array and structure accesses involved run time calculations repeated within inner loops and unnecessary calculations were performed. There are many optimizations traditional compilers perform to minimize this arithmetic. Writing the equivalent program in Forth exposed these excess calculations directly to the programmer. Thus the high level Forth source code would normally be written to avoid these inefficiencies.

Finally, the algorithms and data structures used by the Stanford programs were heavily influenced by traditional languages. A version of one of them, *Towers of Hanoi*, ran 9.6 times faster when coded with data structures and algorithms better suited to Forth than the simple translation of the original code.

The SC32 running with a 10 MHz clock and programmed in Forth was 8.4 times faster on the Stanford benchmarks than a Vax 11/780 programmed in C. If the multiplication dominated *intmm* program is disregarded, then the SC32 is 9.9 times faster. The SC32 is also 19.9 times faster than a 25 MHz Motorola MC68020 running Forth. If the MC68020 is programmed in C than the SC32 is still 1.4 times faster.[3]

Our goal was to develop a processor that could deliver the benefits of an interpreted programming environment without any performance penalty. The data we have collected show that this goal was achieved. Small Forth programs run at least as fast on the SC32 as equivalent C programs on traditional microprocessors. Furthermore it is likely that this relationship will become more favorable for large programs due to the SC32's efficient call/return mechanism.

## 4.5 Applications of the SC32

Several different SC32 based computers have been built at APL. A simple single board computer was designed to demonstrate the chip. That design was later modified and used in telemetry decommutation ground support equipment for the TOPEX and SPINSAT radar altimeter satellites. A standalone computer system, including operating system and utilities, based on magnetic bubble memory for mass storage was developed to show the benefits of self hosted embedded processors for the NASA Goddard Space Flight Center. The most complex SC32 system we have built is a VME bus CPU with full master/slave capability. It will be used to control a balloon borne solar magnetograph. These were interesting projects, but it was not until 1989 that the Freja magnetometer instrument gave us the opportunity to use one of our chips in space flight hardware.

Table 2: Freja Magnetometer Requirements Summary

---

• Anti-alias low pass filters for DC and AC channels
– 64 Hz cutoff during normal rate (14.3 kbits/sec allocated to our instrument) telemetry operations
– 128 Hz cutoff during high rate (28.7 kbits/sec allocated to us) telemetry operations
• Digitize X, Y, Z AC and DC magnetic field measurements to 16 bits
– 128 samples/sec during normal rate telemetry operations
– 256 samples/sec during high rate telemetry operations
• Oversample and average X, Y, and Z DC measurements
• Anti-alias filter one AC channel with 256 Hz cutoff and sample at 512 samples/sec
• Computer amplitude spectrum 0-256 Hz for the AC channel with 512 point FFT
• Detect magnetic activity to trigger data collection in other experiments
• Collect and digitize housekeeping and status data
• Format and output telemetry
• Interpret and execute commands

---

## 5 The Freja Magnetic Field Experiment

Freja is a Swedish satellite that will be launched into a nearly polar orbit to study the earth's magnetosphere and ionosphere. Experiments from Sweden, Germany, and Canada will fly on the satellite and the U.S. is represented by a magnetic field experiment designed and built at APL. Freja is clearly an international effort with launch scheduled in August 1992 as a "piggyback payload" on a People's Republic of China Long March rocket (barring significant changes in the political situation).

### 5.1 Magnetometer Requirements

The magnetometer uses the SC32 to implement the instrument's data acquisition and analysis system. Overall instrument requirements are summarized in Table 2.[4]

The conventional approach to satisfying these requirements would include a switchable hardware anti-aliasing filter (for the two different sample rates), a 16-bit A/D, and an on board computer for status and housekeeping tasks. The processor would be programmed in its assembly language and the code would be cross-assembled on a separate machine. The object code would be downloaded to the target hardware for debugging using in-circuit emulators and other support equipment. No data analysis would be performed on the satellite but would be deferred to ground based postprocessing.

This configuration was not feasible within the resources provided by Freja to our magnetometer. There was neither power nor enough circuit board space for the switchable filters. Filters would also seriously degrade the noise floor of the magnetic field measure-



ments. Telemetry bandwidth precluded transmission of the 512 samples/sec channel to ground for spectral analysis. A separate digital signal processing device used to perform this task would exceed the available power and board space. The extra hardware and software design tasks would also have lengthened our development schedule. Finally, the traditional approach to developing embedded computer software with cross-development tools and in-circuit emulators was too costly due to the long edit, compile, download, and emulate cycle.

Our magnetometer overcame these problems by using a simple fixed hardware anti-aliasing filter, a 16-bit A/D converter, and the SC32 microprocessor. The computer performs data acquisition and averaging, digital anti-alias filtering, FFT computation, telemetry formatting, command interpretation and execution, and other instrument control functions. Software development and debugging were performed interactively on the actual target hardware in a high level language. Despite the processing demands imposed by satisfying these requirements with software, the magnetometer processor has a 50% throughput margin when the SC32 is driven at 40% of its maximum clock rate.

Mass and power requirements were typical of small satellite experiments. The chassis was milled from a solid block of magnesium rather than aluminum and circuit cards were hardwired together instead of using cable assemblies. The completed instrument, excluding probes and boom, weighed 3.5 kg. The entire instrument consumed less than 3.7 W including DC-DC converter, sensor electronics, telemetry subsystem, and the computer itself.

## 5.2 Instrument Development

Schedule and budget constraints were also quite challenging. The flight hardware and software were delivered to Sweden in July 1991, two years after the project was started. We estimate that the hardware and software were developed for 50-75% lower cost than a system of equivalent capability based on a traditional microprocessor such as the 80C86RH. The cost savings were due primarily to our use of an interactive Forth system rather than a cross-compiler/assembler that would be needed for the conventional processor. We also have significant doubt that an equivalent instrument could be based on the 80C86RH due to its limited throughput, even if it were programmed entirely in assembly language.

The productive software development environment provided by the SC32 was a key factor in quickly completing the instrument. Forth's interactive capability greatly assisted hardware debug and subsystem integrations. The flight code was extremely compact, in source (2500 lines) as well as object form (16 Kwords including operating/development system). Small code size was due to two factors. First, our real time scheduler allowed the program to be organized into 8 cooperating tasks. Each task was simple and easily programmed especially when compared to the alternative of a single monolithic piece of code. Secondly, Forth's extensibility meant that program size grew logarithmically as complexity increased. Essentially Forth was used to develop a new programming language specifically oriented to the problem domain. Therefore programs that solved tasks in that domain were very compact. Because of these characteristics of Forth, one of us (Hayes)

was able to write the magnetometer flight software in only two months. The magnetometer was delivered in July 1991 and has since been integrated with the other Freja subsystems. We were the first of the seven experiments on Freja to deliver fully flight-ready hardware and software for satellite integration. No flight software changes have yet been needed.

### 5.3 Radiation Testing

Much of the development of our instrument was affected by considerations of the natural radiation environment in Freja's 600 km x 1700 km high inclination orbit. During the two year Freja mission, we expect to receive a total radiation dose of 12 kRad(Si). Radiation induced latch-up and single event upset (SEU) soft errors also concerned us.

The Freja magnetometer CPU board contains the SC32, two 32 K x 32 RAM modules, two 32 K x 32 EEPROM modules, two 82C54RH timer chips, and 52 SSI/MSI parts. The RAM and EEPROM parts were chosen because other APL flight programs had determined that their radiation characteristics were acceptable in Freja's orbit. The 82C54RH radiation tolerance was guaranteed by its manufacturer. SSI/MSI logic from the 54AC00 family were used for the support chips because they were also known, again due to information from other APL flight projects, to work in our environment. We had to establish the radiation characteristics of the SC32 ourselves.

#### 5.3.1 Total Dose

Two different SC32 fabrication lots were evaluated for total dose characteristics using our in-house Co<sup>60</sup> facility. Exposure was performed at a rate of 1 kRad(Si)/min with bias and a low speed clock applied to force the part into a known state. Bias current was monitored during exposure. Component functionality was assessed within 1-2 min after each radiation exposure using a standalone computer board executing SC32 diagnostics. Testing required no more than five minutes after each exposure step, thus annealing effects were minimized and the entire test was completed within an hour.

The first lot, obtained from our commercial licensee, was fully functional and within parametric limits beyond 15 kRad(Si) for all five parts tested. The mean total dose tolerance of these parts was 19.9 kRad(Si) with a variance of 4.8 kRad(Si). Full functionality returned overnight to all tested parts from this lot after annealing at room temperature with no bias applied.

The other part lot was supplied directly by our foundry and had been packaged according to Mil-Spec-883B. Our reliability group performed a pre-cap visual inspection of these parts at the foundry and found their quality was excellent and that these parts could easily be upgraded to higher reliability levels through APL's in-house testing and screening procedures. Unfortunately, a process change to improve yield in the two years since the first lot had been built degraded total dose tolerance. Three parts from this lot all failed at slightly more than 5 kRad(Si) when tested with same procedures used with the first lot. An additional three parts were exposed to 1 kRad with two days between subsequent exposures to more nearly simulate the radiation environment of the Freja orbit. These

parts also failed at 5 kRad(Si). Room temperature unbiased annealing has only restored functionality to two of these six parts.

Because of the disappointing total dose behavior of the second batch of parts, we were forced to obtain our flight parts from the first lot. Several factors allowed us to upgrade these commercial parts to space flight quality. The positive report on our foundry's quality control was encouraging, both commercial and Mil-Spec parts were packaged in the same high quality ceramic pin-grid array package, and all lots were assembled with the same equipment and personnel at the foundry. So commercial parts from the first lot were extensively screened at APL and passed all tests.

### 5.3.2 Latch-up and SEU

Radiation induced latch-up and SEU sensitivity of the flight part lot were also evaluated. Initially, SC32 parts were screened for latch-up sensitivity in an in-house  $\text{Cf}^{252}$  chamber. This equipment exposed the die to heavy ions with a mean linear energy transfer (LET) of  $36 \text{ Mev-cm}^2/\text{mg}$  at a high flux rate. The SC32 did not latch during a 30 minute exposure. Subsequent work showed that many other chip types also did not latch in the  $\text{Cf}^{252}$  chamber.

However, later tests made at the Single Event Upset Test Facility of the Brookhaven National Laboratory Tandem Van de Graaff accelerator cast doubt on conclusions about latch-up sensitivity based on  $\text{Cf}^{252}$  data. Using the Brookhaven equipment we were able to gather both radiation induced SEU and latch-up sensitivity of the SC32. The chip did latch-up with an LET threshold of  $15.6 \text{ Mev-cm}^2/\text{mg}$  which corresponds to about 1 latch-up per 21 years in the Freja orbit. An SEU threshold of  $5 \text{ Mev-cm}^2/\text{mg}$  was also observed which was estimated to be equivalent to one soft error every 166 days in our orbit.

These radiation testing results led us to add latch-up protection circuitry to the DC-DC converter. If excessive current is drawn by the SC32, the CPU board will be momentarily turned off thus resetting the latched circuitry. After power is restored the computer will resume normal processing.

SEU events are more difficult to detect and their impact can be more subtle. An SEU could disturb the program controlling the processor or it could invalidate a single word of science data. Because an SEU is only expected every few months, it represents only a minor error in the collected data and will be ignored. Program errors will be detected by a watchdog timer that must periodically be updated. An SEU induced program error will most likely be detected by a failure to properly access the watchdog. In response, the watchdog will reboot the system. Both types of radiation induced error should occur rarely enough that these correction strategies will not significantly degrade the quality of the magnetometer data.

## 6 Conclusions

Because the SC32 was originally designed as a research effort and was only manufactured by a commercial foundry, many questions had to be resolved before we could use it a space

based instrument. Reliability concerns were greatly reduced after a site visit to the foundry showed excellent manufacturing procedures were followed. A thorough screening of parts from the flight lot has also added to our confidence in the reliability of the SC32.

Radiation tolerance of our chip was also studied. Early testing of our prototype chips indicated they would meet our needs. Commercial versions of our chip manufactured shortly thereafter were fully evaluated and had acceptable radiation tolerance. However, the foundry modified the manufacturing process to improve yield in the interval between when our prototypes were evaluated and when we ordered Mil-Spec chips for our instrument. This process change had the unfortunate side effect of diminishing total dose tolerance to unacceptable levels. Unless a foundry rigorously controls those aspects of the process that impact radiation tolerance, performance may vary significantly between lots.

We have shown that a Forth language directed microprocessor with hardware and software optimized for embedded systems can significantly improve spacecraft instrumentation. Because of the capabilities of the magnetometer's computer based on the SC32, an instrument of unprecedented capability was developed at far lower cost than could otherwise be achieved.

The most important lesson we have learned from this work is that a custom integrated circuit of the right architecture can deliver substantial benefits even when only one chip is needed. System performance that is unreachable with catalog components can be achieved and qualification issues can be resolved. Most surprisingly, system development costs can be reduced by using custom chips. Savings from designing fewer circuit boards, consuming less power, buying fewer expensive flight components, and most importantly greater software productivity easily balance the additional costs of developing and qualifying the right custom integrated circuit.

## 7 Acknowledgements

In addition to the authors, Susan Lee, Susan Waters, Mary Wong, and Tom Zaremba have all contributed to this work over the past decade. We greatly appreciate the advice and assistance received from many people in APL's SOR group in making our instruments as reliable as possible. The many talents our shop groups contributed to this work were also vital to our success. Finally, we would like to thank Larry Zanetti, the Freja magnetometer's principal investigator, for encouraging us to use the SC32 in his instrument, and our management, particularly Jay Dettmer and Tom Zaremba, for their support and encouragement of our efforts.

## References

- [1] B. Ballard and J. Hayes, Forth and Space at the Applied Physics Laboratory, *Proc. of the 1991 Rochester Forth Conference*, Inst. of App. Forth, Rochester, N.Y., June 1991.

- [2] J. Hayes and S. Lee, The Architecture of the SC32 Forth Engine, *J. of Forth App. and Res.*, V5, N4, pp. 493-506.
- [3] M. Fraeman, Performance Evaluation of the SC32 Stack Microprocessor, *Proc. of the 1989 Rochester Forth Conference*, Inst. of App. Forth, Rochester, N.Y., June 1989.
- [4] R. Henshaw, B. Ballard, J. Hayes, and D. A. Lohr, An Innovative On-Board Processor for Lightsats, *Proc. of the 4<sup>th</sup> AIAA/USU Conf. on Small Satellites*, AIAA, August 1990.



## **Multi-chip Modules: A High-performance Packaging Alternative**

L. Salmon  
Brigham Young University

*Abstract-* Multi-chip Module (MCM) packaging has emerged as an important technology for high-performance electronic systems. Benefits of MCMs include: high IC packing density, low interconnect propagation delay, excellent power dissipation characteristics, and low cost. This paper will review MCM substrate fabrication, testing, and design. Major challenges for MCM implementation in high-performance systems will be discussed. Finally, applications of MCM technology to current high-end computer systems will be reviewed.





# New Dynamic FET Logic and Serial Memory Circuits for VLSI GaAs Technology

A. G. Eldin

Electrical Engineering Department,  
The University of Calgary,  
Calgary, Alberta, Canada.

**Abstract-** The complexity of GaAs FET VLSI circuits is limited by the maximum power dissipation while the uniformity of the device parameters determines the functional yield. In this work, novel digital GaAs FET circuits are presented that eliminate the dc power dissipation, reduce the area to 50% of that of the conventional static circuits and its larger tolerance to device parameters variations, results in higher functional yield.

## 1 Introduction

GaAs technology is used in the fabrication of ultra fast digital integrated circuits. The availability of such circuits is critical for many applications such as Gigabit communication systems and super fast computers [1]. The GaAs FET is fundamentally different from both the MOSFET and the bipolar transistor. Figure 1 highlights these differences.

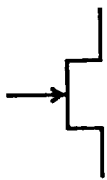
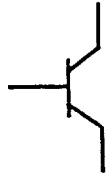

		
<input type="checkbox"/> GaAs FET	<input type="checkbox"/> Bipolar	<input type="checkbox"/> MOSFET
<input type="checkbox"/> Voltage driven	<input type="checkbox"/> Current driven	<input type="checkbox"/> Voltage driven
<input type="checkbox"/> Low $R_{in}$	<input type="checkbox"/> Low $R_{in}$	<input type="checkbox"/> High $R_{in}$
<input type="checkbox"/> $V_{in}$ is Clamped to (0.6 - 0.7) volts $V_{in}$ is the driving signal	<input type="checkbox"/> $V_{in}$ is Clamped But $I_B$ is the driving signal	<input type="checkbox"/> $V_{in}$ is limited by the gate oxide breakdown voltage

Figure 1: Fundamental differences between devices

Because of these differences, the bipolar and MOS logic families and circuit techniques

### 1.3.2

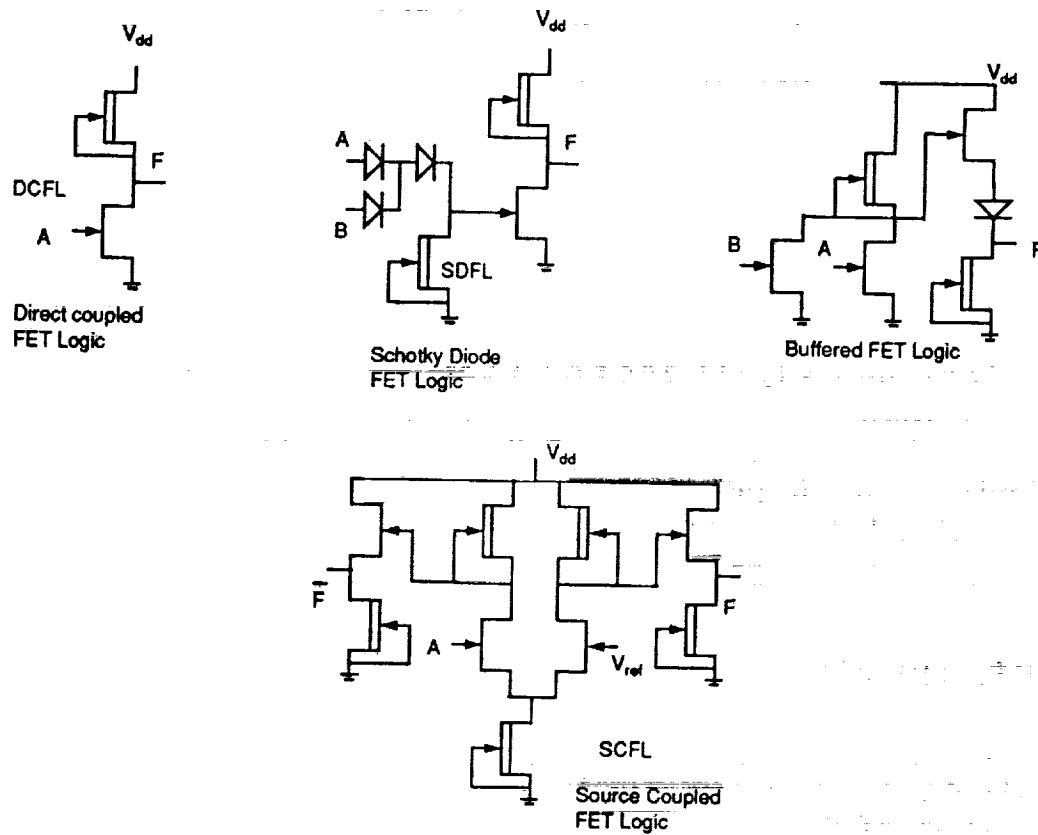


Figure 2: Static GaAs FET logic families

Because of these differences, the bipolar and MOS logic families and circuit techniques will be less successful if applied directly without modification to the GaAs technology. Figure 2 shows the most commonly used static GaAs FET logic families [1].

In any of the two logic state, all these circuits will dissipate dc power. This dc component accounts for 90% of the total power [2]. The power dissipation limits the maximum number of gates to 15,000 assuming a maximum allowable chip power of 5 watts [2]. On the other hand, the ratio of the threshold voltage variation to the noise margin is critical for determining the IC electrical yield. It is shown that if the threshold voltage variance is changed from 90 mv to 150 mv, the circuit size should be reduced from 10,000 to 100 gates to maintain 50% yield [2]. This illustrates that the threshold voltage must be tightly controlled for acceptable yield. In this paper, a novel circuit technique [3]-[6] is applied to GaAs HFET technology to overcome these two main limitations. The new memory and logic circuits do not dissipate any dc power, are less sensitive to threshold voltage variation and have very small size.

## 2 The D-Type Flip Flop

An intermediate stage of a dynamic shift register is shown on Figure 3 a. The D-type flip flop uses depletion type transistors with threshold voltage of (-0.7 volts). Figure 3 b shows the clock and input waveforms and the logic levels.

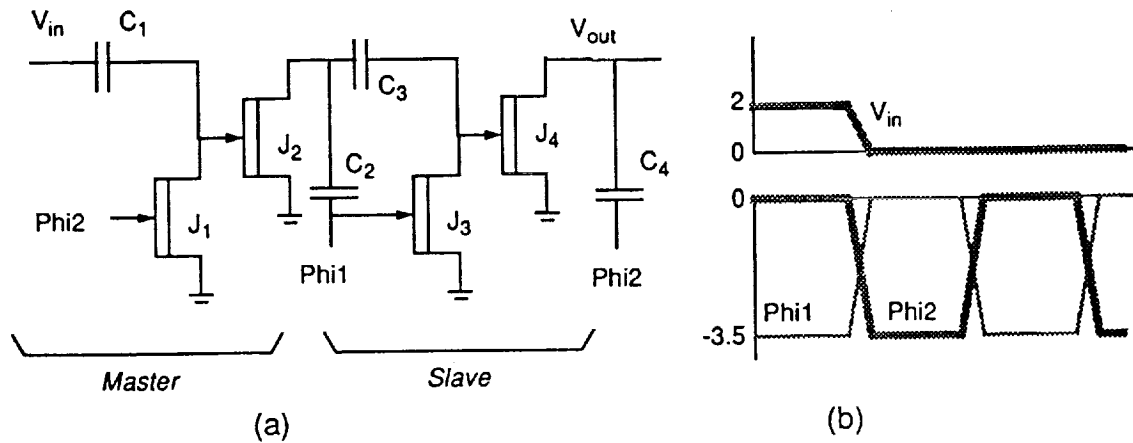


Figure 3: M/S dynamic D-type Flip Flop (*intermediate stage*)

During  $t_1$ ,  $\Phi_{i2} = 0$  volts. The master section is in the sample phase. The input data is stored on the capacitor  $C_1$ . It is charged to 2 volts or discharged to 0 volts for  $V_{in}$  being logical 1 or 0 respectively. The capacitor  $C_2$  is precharged to approximately 3.5 volts through  $J_2$  ( $\Phi_{i1} = -3.5$  volts). The slave section is in the evaluation phase. Transistor  $J_3$  is cut off and the drain voltage of  $J_2$ ,  $V_{D2} = 0$  volts, thus providing a reference voltage for evaluating the stored data on  $C_3$ . If  $C_3$  is charged,  $J_4$  is turned off and the precharged capacitor  $C_4$  retains its voltage to represent logic 1. However, if  $C_3$  is discharged,  $J_4$  is turned on and  $C_4$  is discharged to represent logic 0. During  $t_2$ , the roles of the master and slave sections are interchanged. Figure 4 shows the output stage of the shift register.

The capacitor  $C_4$  is replaced by a pull up device for interfacing with static logic (DCFL). The simulation results for the output stage are shown in Figure 5. The device model accounts for the second order effects and is accurately calibrated to a 1  $\mu\text{m}$  HFET process. Figure 5(a) shows that  $V_{out}$  is delayed by one clock period with respect to  $V_{in}$  which verifies the operation of the D-Type flip flop at 2 GHz. Figure 5(b) shows the waveforms  $V_{D1}$ ,  $V_{D2}$  and  $V_{D3}$  which correspond to the drain voltage of  $J_1$ ,  $J_2$  and  $J_3$  respectively.

Table 1 compares the dynamic and static (DCFL) implementations of the D-Type flip flop.

Each section of the DCFL (M/S) Flip Flop uses two inverters for the static memory cell, two depletion transistors (clocked transmission gates) and two DCFL super buffers, as shown in Figure 6, to properly buffer the memory cell from the direct and capacitive coupling caused by the clock signals driving the transmission gates. This DCFL implementation requires 24 transistors of both depletion and enhancement types. Table 1 shows

### 1.3.4

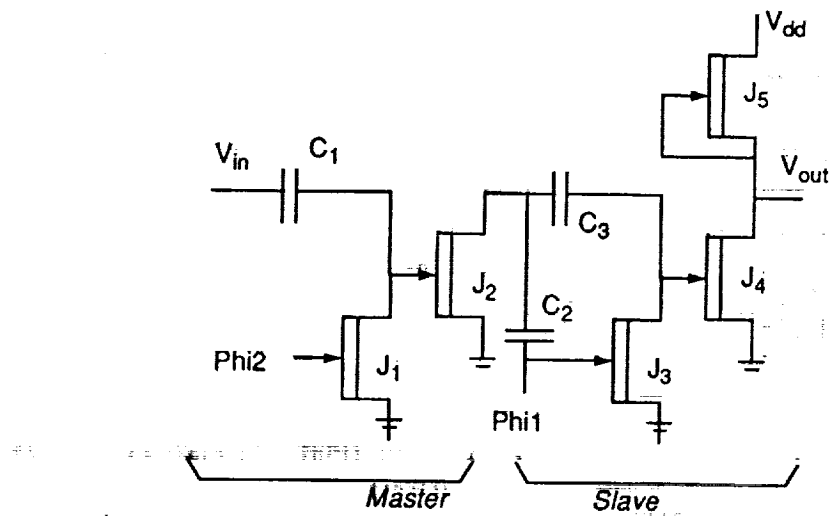
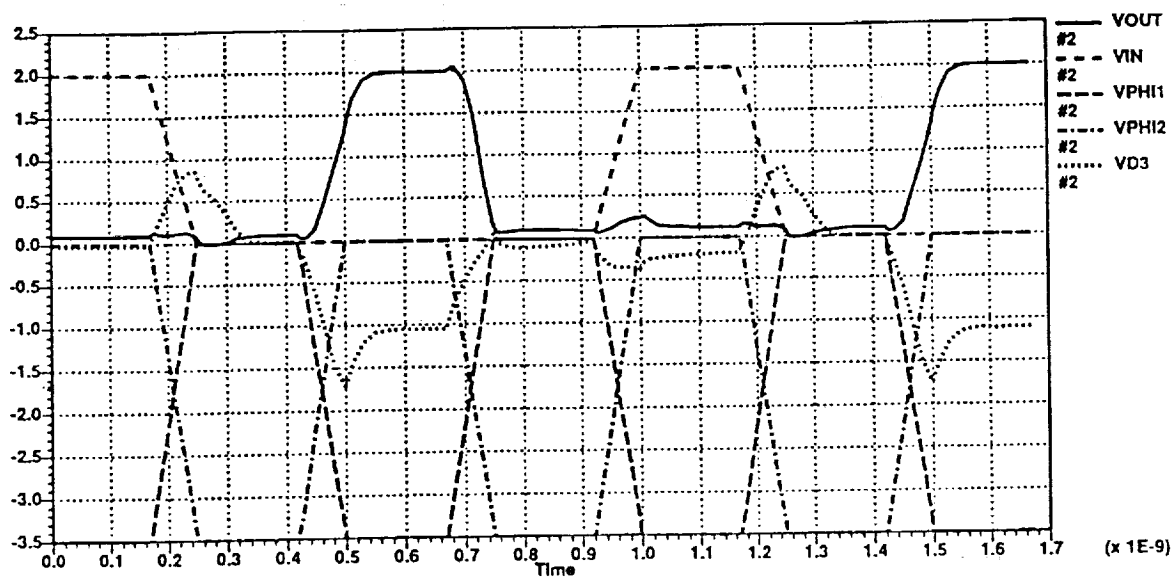


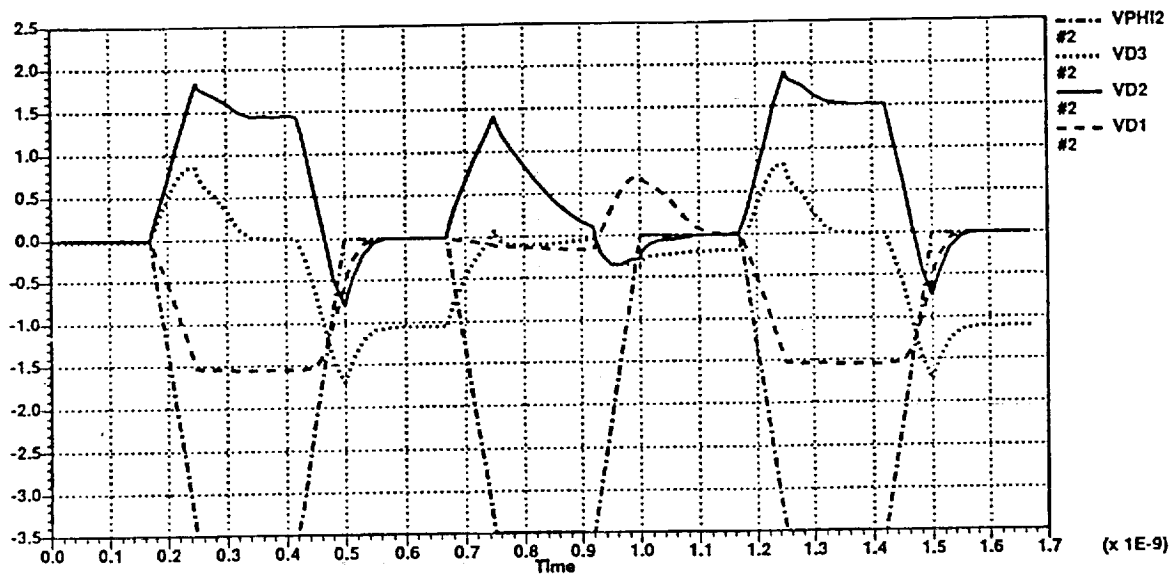
Figure 4: M/S dynamic D-type Flip Flop (*Output stage*)

	Dynamic FF	Static FF
Power	0.4 mW	4 mW
Number of devices	4 transistors 4 capacitors	24 transistors
Relative area	0.3	1
Noise margin (NM)	500 mV	200 mV

Table 1: Dynamic and static implementations of the D-Type flip flop



(a)



(b)

Figure 5: Voltage waveforms of the M/S D-Type Flip Flop

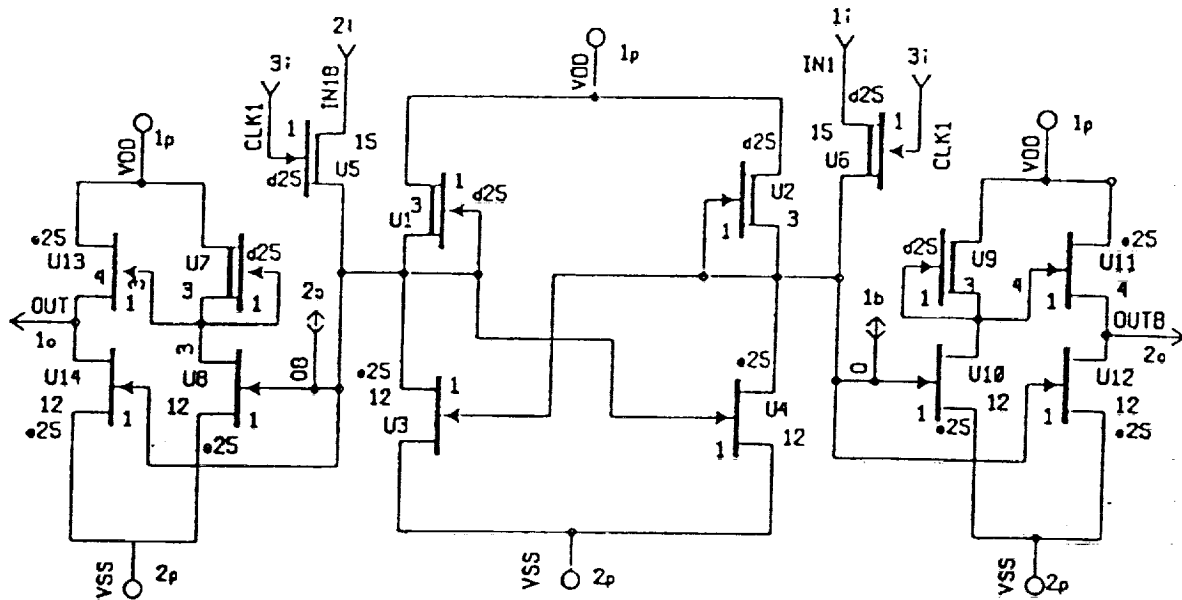


Figure 6: DCFL implementation of one section of the M/S D-Type Flip Flop

that the dynamic circuit has significant savings in both power dissipation and area. The larger functional yield can be measured by the ratio of the threshold voltage variation to the noise margin. Figure 5 shows that  $V_{D1}$  swings between -1.2 and 0 volts with a threshold voltage of -0.7 volts. This relatively large noise margin makes the circuit operation less sensitive to threshold voltage variations and results in a larger functional yield.

### 3 Logic circuits implementation

The basic dynamic circuit can also be used to implement the AND, OR, and complex logic functions. The operation of the basic circuit is similar to that of the dynamic flip flop. Figure 7 summarizes the operation.

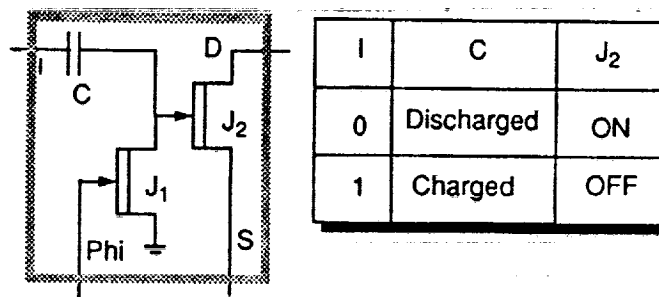


Figure 7: Basic dynamic logic circuit

A	B	C <sub>1</sub>	C <sub>2</sub>	J <sub>1</sub>	J <sub>2</sub>	F
1	1	charged	charged	OFF	OFF	1

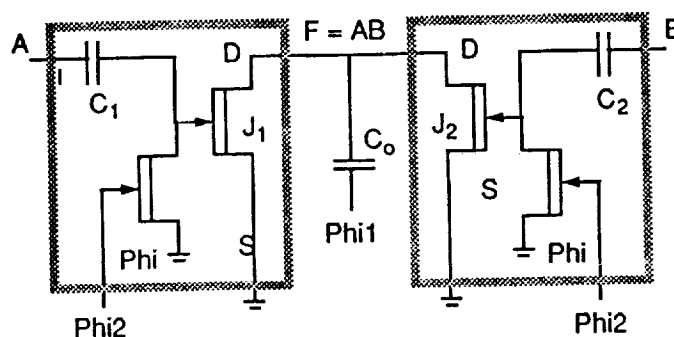


Figure 8: The dynamic AND function

When the input is logic 0, the capacitor  $C$  is discharged during the sampling phase and the transistor  $J_2$  will turn on during the evaluation phase. Similarly, if the input is logical 1, the capacitor is charged during the sampling phase causing  $J_2$  to turn off during the evaluation phase.

An AND function is realized by connecting two cells in parallel as shown in Figure 8. During the evaluation phase, the output will remain charged (representing logic 1) if both  $J_1$  and  $J_2$  are turned off. This corresponds to  $A = B = 1$  during the sampling phase. Any other combination for the values of the inputs  $A$  and  $B$  will result in at least one of  $J_1$  or  $J_2$  being turned on and causing the output to correspond to logic 0.

The OR function is implemented by connecting cells in series as shown in Figure 9. During the evaluation phase, the output will be discharged only if both  $J_1$  and  $J_2$  are turned on. This corresponds to  $A = B = 1$  during the sampling phase. If  $A$  or  $B$  is logic 1, at least one of the transistors  $J_1$  or  $J_2$  will be turned off during the evaluation phase. This causes the output to remain charged and correspond to logic 1.

Complex logic gates can be realized by parallel and series connections of the basic circuit as shown in Figure 10. In this example the output  $F$  will remain charged during the evaluation phase if either ( $C$  and  $D$ ) are logic 1 during the sampling phase or ( $A$  and  $B$ ) are logic 1. This ensures that there is no discharge path from the output to ground during the evaluation phase.

It is seen that these logic circuits do not dissipate any dc power. Also, since only one type (depletion) of transistors is used, the circuits are less sensitive to process and threshold voltage variations. It is noted that when the clocked depletion transistors are turned on,  $V_{GS} = 0$  volts and the transistors do not draw any gate current.  $V_{GS}$  can be increased to about 0.4 volts, which will keep the gate current negligibly small while increasing the driving capability of the clocked transistors and the noise margin of the circuit. This will also enhance the operating speed.

A	B	C <sub>1</sub>	C <sub>2</sub>	J <sub>1</sub>	J <sub>2</sub>	F
0	0	discharged	discharged	ON	ON	0

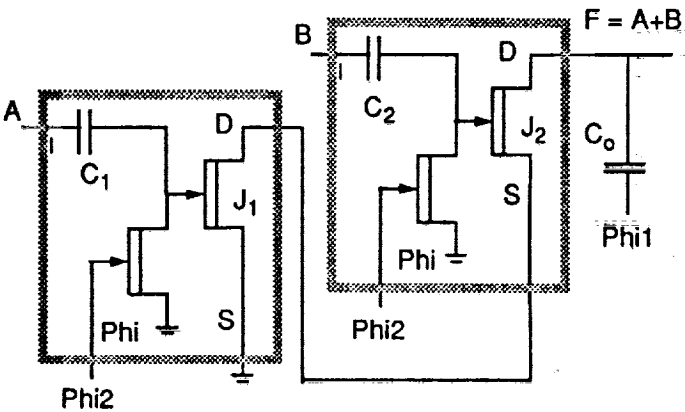


Figure 9: The dynamic OR function

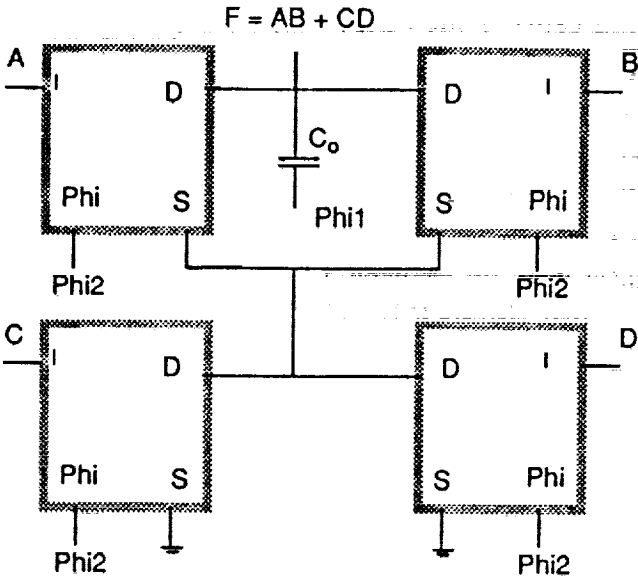


Figure 10: The dynamic Complex Logic gate



## 4 Conclusions

The new dynamic circuits eliminate the dc power and have large noise margin and small size. Compared to the static implementation using the DCFL, the use of the dynamic circuits results in (50-70)% reduction in the area. The noise margin and therefore the electrical functional yield is increased by a factor of 2.5 and the total power dissipation is reduced by 90% at a switching speed of 2 GHZ. These significant improvements allow an order of magnitude higher level of integration with acceptable functional yield and power dissipation.

## References

- [1] S. Long and S. Butner, Gallium Arsenide Digital Integrated Circuit Design, McGraw Hill Publishing Company, 1990.
- [2] S. Long and M. Sundaram, Noise-Margin Limitations on Gallium-Arsenide VLSI, *IEEE J. Solid State Circuits*, Vol. 23, pp. 893-900, August 1988.
- [3] A. Eldin and M. Elmasry, VLSI Dynamic Memory, United States Patent # 4,791,611 Dec. 13, 1988.
- [4] A. Eldin and M. Elmasry, New Dynamic Logic and Memory Circuit Structures For BICMOS Technologies, *IEEE Journal of Solid State Circuits*, Vol. SC-22, pp. 450-453, June 1987.
- [5] A. Eldin and M. Elmasry, New Dynamic Logic and Memory Circuit Structures For BICMOS Technologies, *Twelfth European Solid State Circuits Conference*, Tech. Dig., pp. 4-6, Delft, The Netherlands, September 1986.
- [6] A. Eldin and M. Elmasry, A Novel JCMOS Dynamic RAM Cell For VLSI Memories, *IEEE Journal of Solid State Circuits*, Vol. SC-15, pp. 715-723, June 1985.

1. The first part of the document is a letter from the President of the United States to the Congress, dated January 3, 1862. It is a very important document, as it contains the President's annual message to Congress. The letter is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

2. The second part of the document is a report from the Secretary of the Treasury, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

3. The third part of the document is a report from the Secretary of the Interior, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

4. The fourth part of the document is a report from the Secretary of the War, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

5. The fifth part of the document is a report from the Secretary of the Navy, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

6. The sixth part of the document is a report from the Secretary of the State, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

7. The seventh part of the document is a report from the Secretary of the War, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

8. The eighth part of the document is a report from the Secretary of the Navy, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

9. The ninth part of the document is a report from the Secretary of the State, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

10. The tenth part of the document is a report from the Secretary of the War, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress. The report is written in a formal, dignified style, and it is one of the most important documents in the history of the United States.

## Automated ILA Design for Synchronous Sequential Circuits

M. N. Liu, K. Z. Liu, G. K. Maki and S. R. Whitaker  
NASA Space Engineering Research Center for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract** - This paper presents an ILA architecture for synchronous sequential circuits. This technique utilizes linear algebra to produce the design equations. The ILA realization of synchronous sequential logic can be fully automated with a computer program. A programmable design procedure is proposed to fulfill the design task and layout generation. A software algorithm in the C language has been developed and tested to generate 1  $\mu$ m CMOS layouts using the Hewlett-Packard FUNGEN module generator shell.

### 1 Introduction

The design of sequential circuits presents a major task for most digital systems. As Very Large Scale Integrated (VLSI) technology advances, developing an architecture to maximize the efficiencies of all the design steps becomes a major goal in the research of sequential circuit design.

This paper introduces the Iterative Logic Array (ILA) as a new architecture for synchronous sequential circuits. This architecture realizes a sequential circuit by replicating simple basic modules. With an ILA architecture, a sequential machine can be built into a very regular form automatically by a computer program with a single type of ILA module. The simplicity and programmability of the ILA architecture significantly reduce the design task in all stages of VLSI implementation, from logic design, circuit design, artwork generation to verification.

### 2 ILA Architecture

Iterative Logic Arrays (ILA) have been described in the literature for quite some time [1,2]. An ILA circuit consists of an array of identical cells. Generally, as shown in Figure 1, each ILA cell contains two sets of input signals. One set of inputs are applied in parallel, while the other set of inputs are driven by adjacent cells. Signals normally propagate in only one direction between cells, and outputs are derived only from the serial outputs of the last cell.

In an ILA architecture for sequential circuits, the next state of each state variable is generated by a slice of concatenated ILA cells. A sequential network is then constructed by placing the ILA slices side by side.

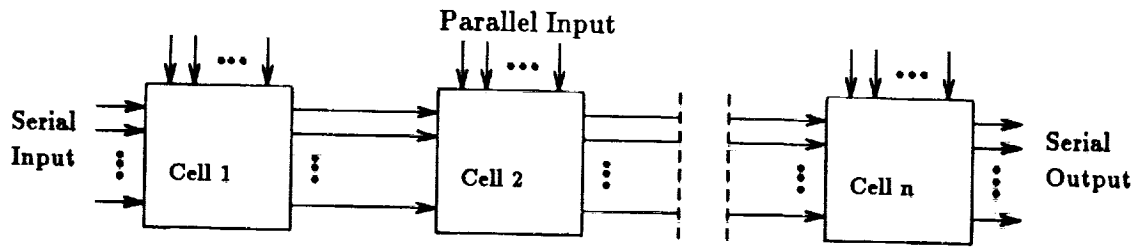


Figure 1: A slice of ILA circuit

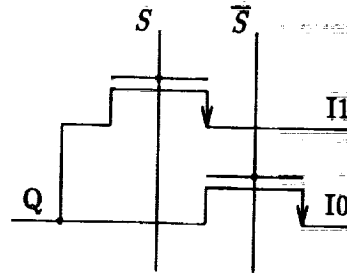


Figure 2: Pass transistor 2-to-1 MUX

The basic cell of an ILA sequential network consists of a 2-to-1 multiplexer (MUX) and a next state forming logic. A MUX cell has a select line  $S$ , its complement  $\bar{S}$  and two data inputs  $I_0$  and  $I_1$ , and a logic function defined by Equation 1.

$$Q = S * I_1 + \bar{S} * I_0 \quad (1)$$

The simplest way to implement the MUX function is to use a pass transistor circuit. Basically, the pass transistor MUX, excluding level restoration logic, is a module of two pass transistors, which functions as two simple switches. Figure 2 shows the circuit of two inputs  $I_1$  and  $I_0$  and one output  $Q$  controlled by two control lines  $S$  and  $\bar{S}$  which are assumed to be asserted exclusively such that only one of two inputs  $I_1$  and  $I_0$  can be passed to  $Q$  at a given time.

Some details in pass transistor transmission characteristics are omitted here. Design considerations, such as level restoration, are assumed to be handled by the output buffers. The circuit design considerations have been discussed in [3,4,5].

### 3 Operational Function

In this research, the one-hot-code is utilized as the state assignment for a synchronous flow table. With the one-hot-code assignment, there is a unique state variable corresponding to each state. That makes it possible to express the design function using the states in the flow table explicitly. A new form of mathematical expression is proposed next which describes a flow table directly by flow table states.

**Definition 1** *The set of operational functions is the behavior description of a synchronous flow table of  $n$  rows and  $m$  columns. Each function is an equation for a next state  $\hat{S}_i$  in the flow table.*

$$\hat{S}_i = \sum_{p=1}^m s_{ip} I_p \quad (2)$$

where  $s_{ip}$  is an OR function of the states  $S_j, \forall j = 1, \dots, n$ , which have  $S_i$  as the next entry under input  $I_p$ .

It can be shown that there is a one-to-one mapping between the next state equation

$$Y_i = \sum_{p=1}^m f_{ip} I_p \quad (3)$$

and the operational function. With the one-hot-code state assignment, each  $\tau$ -partition can be expressed as

$$\tau = \{S_i; S\}$$

which partitions a single state  $S_i$  from the rest states  $S$  in the flow table. The number of state variables is equal to the number of states. Next state  $\eta$ -partitions can be formed using known procedures [6]. If an  $\eta$ -partition  $\eta_i$  is

$$\eta_i = \{S_1 S_2 \dots S_i; S\}$$

then it is well known that

$$f_{ip} = y_1 + y_2 + \dots + y_i.$$

On the other hand, Equation 2

$$\hat{S}_i = \sum_{p=1}^m s_{ip} I_p$$

can be mapped into a next state equation as Equation 3 if the one-hot-code assignment is used where  $f_{ip}$  are sum of the state variables  $y_j$  corresponding to  $s_{ip}$  in Equation 2. Therefore, there is a one-to-one mapping between Equation 2 and Equation 3.

Since the operational function is a direct representation of the flow table, they can be derived by inspection. For each state in the next state entry, there is a product term of the present state and input state in the operational function. If a synchronous machine is specified by a state diagram, the state diagram may need to be converted to a flow table, though it will not be too hard for an experienced designer to derive the operational functions from the state diagram directly.

Table 1 is the flow table of a state machine with four states. For example, State  $S_a$  appears as the next state entry of states  $S_b, S_c$  and  $S_d$  under  $I_1$ . Therefore, the operational function for  $S_a$  is

$$\hat{S}_a = (S_b + S_c + S_d)I_1.$$

For State  $S_b$ , it appears as the next state under both  $I_1$  and  $I_2$ . So the operational function for state  $S_b$  is

$$\hat{S}_b = S_a I_1 + (S_b + S_c)I_2.$$

	$I_1$	$I_2$	$I_3$
$S_a$	$\bar{S}_b$	$\bar{S}_d$	$\bar{S}_c$
$S_b$	$S_a$	$S_b$	$S_c$
$S_c$	$S_a$	$S_b$	$S_d$
$S_d$	$S_a$	$S_d$	$S_d$

Table 1: A synchronous flow table for the state diagram

The operational functions for state  $S_c$  and  $S_d$  can be derived in the same way. All together, the operational functions for Table 1 are as follows:

$$\begin{aligned}
 \hat{S}_a &= (S_b + S_c + S_d)I_1 \\
 \hat{S}_b &= S_a I_1 + (S_b + S_c)I_2 \\
 \hat{S}_c &= (S_a + S_b + \bar{S}_d)I_3 \\
 \hat{S}_d &= (S_a + S_d)I_2 + (S_c + S_d)I_3
 \end{aligned} \tag{4}$$

## 4 ILA Architecture for Synchronous Sequential Circuits

A simple regular ILA structure requires:

- The design equation is convertible to a pass logic function where each control variable passes a single pass variable or a constant.
- The control variables are shared with each pass logic function.

With such a structure, if the pass variables in each equation are the same, the signal bus to each slice of ILA circuit can be minimized to a single wire.

If state  $S_i$  is used as the control and  $S_i$  appears as a next state under only one input  $I_p$ , then  $I_p$  can be the only pass variable in the design equation for  $\hat{S}_i$ . For example, the equation for  $\hat{S}_a$  in Equation 4 can be converted into a pass logic function with input  $I_1$  as a pass variable:

$$\hat{S}_a = \bar{S}_b(I_1) + \bar{S}_d(I_1) + \bar{S}_c(I_1)$$

From the definition of the operational function in Equation 2, if  $S_i$  appears only under input  $I_p$ , then Equation 2 can be rewritten as:

$$\hat{S}_i = s_{ip} I_p \tag{5}$$

where  $s_{ip}$  is an OR function of the states  $S_k, k \in \{1, 2, \dots, n\}$ . Therefore, Equation 5 can be written into:

$$\hat{S}_i = I_p \sum_{k=1}^n g_{ik} S_k \tag{6}$$



	$I_1$	$I_2$	$I_3$
$S_a$	$S_{b1}$	$S_{d2}$	$S_c$
$S_{b1}$	$S_a$	$S_{b2}$	$S_c$
$S_{b2}$	$S_a$	$S_{b2}$	$S_c$
$S_c$	$S_a$	$S_{b2}$	$S_{d3}$
$S_{d2}$	$S_a$	$S_{d2}$	$S_{d3}$
$S_{d3}$	$S_a$	$S_{d2}$	$S_{d3}$

Table 2: A revised flow table

All other operational functions are also in the same form. The results are shown as follows:

$$\begin{aligned}
 \hat{S}_a &= 0 + S_{b1}I_1 + S_{b2}I_1 + S_cI_1 + S_{d2}I_1 + S_{d3}I_1 \\
 \hat{S}_{b1} &= S_aI_1 + 0 + 0 + 0 + 0 + 0 \\
 \hat{S}_{b2} &= 0 + S_{b1}I_2 + S_{b2}I_2 + S_cI_2 + 0 + 0 \\
 \hat{S}_c &= S_aI_3 + S_{b1}I_3 + S_{b2}I_3 + 0 + 0 + 0 \\
 \hat{S}_{d2} &= S_aI_2 + 0 + 0 + 0 + S_{d2}I_2 + S_{d3}I_2 \\
 \hat{S}_{d3} &= 0 + 0 + 0 + S_cI_3 + S_{d2}I_3 + S_{d3}I_3
 \end{aligned}$$

Splitting states in a flow table allows all of the pass variables in an operational function to be the same. The disadvantage of splitting states is that it generates additional next state equations. Increasing the number of equations implies increasing the area in silicon. It is a trade off by gaining programmability and regularity of the ILA realization versus cost. An automated sequential circuit design will significantly reduce the design effort and speedup the process of implementation.

## 5 The Matrix Expression

The operational functions discussed in previous sections can be efficiently expressed with matrices. The matrix will also help to implement the function in silicon. With Equation 6, a synchronous sequential circuit can be expressed with a set of equations:

$$\hat{S}_1 = I_p \sum_{k=1}^n g_{1k} S_k$$

...

$$\hat{S}_n = I_q \sum_{k=1}^n g_{nk} S_k$$

Such a set of equations are equivalent to a matrix expression:

$$\hat{\mathbf{S}} = \mathbf{A} \times \mathbf{G} \times \mathbf{S} \quad (8)$$



where matrices  $\hat{\mathbf{S}}$  are  $\mathbf{S}$  are column vectors

$$\hat{\mathbf{S}} \equiv \begin{pmatrix} \hat{S}_1 \\ \hat{S}_2 \\ \vdots \\ \hat{S}_n \end{pmatrix}; \quad \mathbf{S} \equiv \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{pmatrix};$$

matrix  $\mathbf{A}$  is a diagonal matrix with  $I_p$  in the  $i^{th}$  row/column if the next state  $S_i$  is under  $I_p$ ,

$$\mathbf{A} \equiv \begin{pmatrix} I_i & & & & 0 \\ & \ddots & & & \\ & & \ddots & & \\ 0 & & & \ddots & \\ & & & & I_j \end{pmatrix}$$

and matrix  $\mathbf{G}$  is defined as

$$\mathbf{G} \equiv \begin{pmatrix} g_{11} & \cdot & \cdot & \cdot & \cdot & g_{1n} \\ \cdot & \cdot & & & & \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ g_{n1} & & & & & g_{nn} \end{pmatrix}$$

in which

$$g_{ik} = \begin{cases} 1 & \text{if } S_i \text{ is the next state of } S_k \\ 0 & \text{if } S_i \text{ is not the next state of } S_k \end{cases}$$

For example, the matrix expression for the flow table in Table 2 is:

$$\begin{pmatrix} \hat{S}_a \\ \hat{S}_{b1} \\ \hat{S}_{b2} \\ \hat{S}_c \\ \hat{S}_{d1} \\ \hat{S}_{d2} \end{pmatrix} = \begin{pmatrix} I_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & I_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & I_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & I_3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} S_a \\ S_{b1} \\ S_{b2} \\ S_c \\ S_{d1} \\ S_{d2} \end{pmatrix} \quad (9)$$

The matrix  $\mathbf{A}$  and  $\mathbf{G}$  are directly related to hardware structure. As in the ILA realization, there will be a slice of the ILA circuit for each design equation, as shown in Figure 3. Now each element at the diagonal of the matrix  $\mathbf{A}$  indicates the input state to the ILA slice. Each row of matrix  $\mathbf{G}$  reveals the location of ILA cells in the slice. If the element  $g_{ik}$  is 1, then an ILA cell will be placed under the control of state  $S_k$  in the slice of the ILA circuit for next state  $\hat{S}_i$ . If  $g_{ik}$  is equal to 0, a wire will be placed in that position. An example of the ILA realization will be shown in next section.

## 6 Design Procedure

From the discussion in the previous section, the design of a synchronous machine can be completely automated by programming an ILA cell or a wire into a pre-interconnected layout floor. It allows the physical layout to be designed and stored in computer as a set of building blocks. Then for each instance of synchronous sequential logic, an ILA circuit can be implemented by placing ILA cells according to the corresponding  $G$  matrix. The  $A$  matrix will indicate the interconnection to input states.

From the layout point of view, a wire can be considered as a cell as well. Hence there will be two cell types in an ILA realization. Let the ILA cell which performs the multiplexer function be defined as a *MUX cell*. Let a wire be defined as an ILA *ZERO cell*. Now the computer program can search the  $G$  matrix and place a MUX cell once a "1" is encountered or a ZERO cell once a "0" is encountered. The schematic of a MUX cell and a ZERO cell are shown in Figure 4 (b) and (c) respectively.

### Procedure 1 Synchronous ILA network design procedure.

- Step 1. For a synchronous machine specified by a state diagram, convert it into a synchronous flow table (state table).
- Step 2. If a state appears as a next state under more than one input column, split the state and give a unique name to the state under each input column. Repeat this step until all states under one column are distinguished from states under other columns.
- Step 3. Generate the  $A$  matrix by setting the diagonal element in the  $i^{\text{th}}$  column to be  $I_p$  if state  $S_i$  appears as a next state in the flow table under  $I_p$ .
- Step 4. Generate the  $G$  matrix such that  $g_{ij}$  is "1" if  $S_i$  is the next state of  $S_j$ ,  $g_{ij} = 0$  otherwise.
- Step 5. Map the matrices to the layout floor. Place a MUX cell under the control of  $S_k$  in the slice of the ILA circuit for the next state  $\hat{S}_i$  if  $g_{ik} = 1$  or place a ZERO cell if  $g_{ik} = 0$ .
- Step 6. Connect Input-1 of the last ILA cell in the slice of the ILA circuit for  $\hat{S}_i$  to  $I_p$  which is the diagonal element of matrix  $A$  in the  $i^{\text{th}}$  row. Connect Input-0 of the last ILA cell to the level of logic low (VSS).

For example, for a synchronous machine specified a flow table shown in Table 1, it needs to find those states which are under more than one input state and to split them. The result of splitting is shown in Table 2. The matrices of the flow table can then be generated. For instance,  $S_a$  is a next state under  $I_1$  of state  $S_{b1}$ ,  $S_{b2}$ ,  $S_c$ ,  $S_{d2}$  and  $S_{d3}$ . Then  $I_1$  becomes the diagonal element  $a_{11}$  in matrix  $A$ ; the 1<sup>st</sup> row of matrix  $G$  will have a "0" in the first column since the next state of  $S_a$  under  $I_1$  is not  $S_a$ , and have a "1" in the rest of columns. The  $A$  matrix and  $G$  matrix can then be mapped into an ILA network. The result is shown in Figure 4 (a) where each ILA cell is represented by a box. The boxes in dash line represent the ZERO cell ( $g_{ij} = 0$ ) and boxes in solid line represent the MUX cell

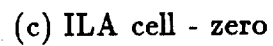
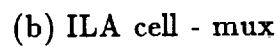
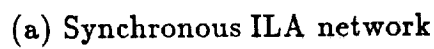


Figure 4: The ILA network for the example

( $g_{ij} = 1$ ). As the first row of matrix  $\mathbf{G}$  is "011111", the top slice of the ILA for  $\hat{S}_a$  consists of one ZERO cell on the left and five MUX cells. Again, from matrix  $\mathbf{A}$ , the input of the last ILA cell is tied to  $I_1$  and VSS.

As mentioned before, a major advantage of the design approach in Procedure 1 is that it allows a hierarchical layout design. The high level layout, including interconnections, is identical for all synchronous flow tables. When the function of a flow table changes, the only thing one has to do is to instruct the computer to re-program the position of MUX cell and ZERO cell. Of course, the input state to each slice of the ILA may need to be changed as well.

## 7 Automated Synchronous ILA Design System

The ILA design procedure has been coded into computer programs and ported to Hewlett-Packard FUNGEN layout tool. The automatic synchronous ILA design system consists of an HP FUNGEN shell and three major subsystems:

- Sequential Logic Processor
- FUNGEN Configuration Code
- Library of Layout Building Blocks.

The Sequential Logic Processor is an ILA circuit topology generator which receives the specification of synchronous sequential machine and converts it into a form specified by FUNGEN Configuration Code. There are three phases in implementing the Sequential Logic Processor: *flow table revising*, *matrices generation* and *FGNRC formation*. The first two phases follows closely to the step 2, step 3 and step 4 in Procedure 1. The third phase is to generate parameters of device modules pre-defined by FUNGEN Configuration Code and write them into a FGNRC file. By modifying the last phase, the program can be ported to any other artwork generator systems.

The FUNGEN Configuration Code describes the artwork architecture and defines the modules in the FGNRC file. The FUNGEN Configuration Code is written in Fungen Configuration Language (FCL), a subset of C language with a number of functions for Hewlett-Packard TRANTOR database generation. The overall ILA architecture and a set of ILA configuration modules are specified in the FUNGEN Configuration Code.

When running the FUNGEN shell, the system invokes the FUNGEN Configuration Code, FGNRC file and Layout Library, and automatically generates a layout artwork by placing pre-designed ILA cells and peripheral buffers. It also labels all of blocks in accordance with the FUNGEN Configuration Code and FGNRC file. Figure 5 illustrates the block diagram of the ILA design system and the algorithm of Sequential Logic Processor implementation.

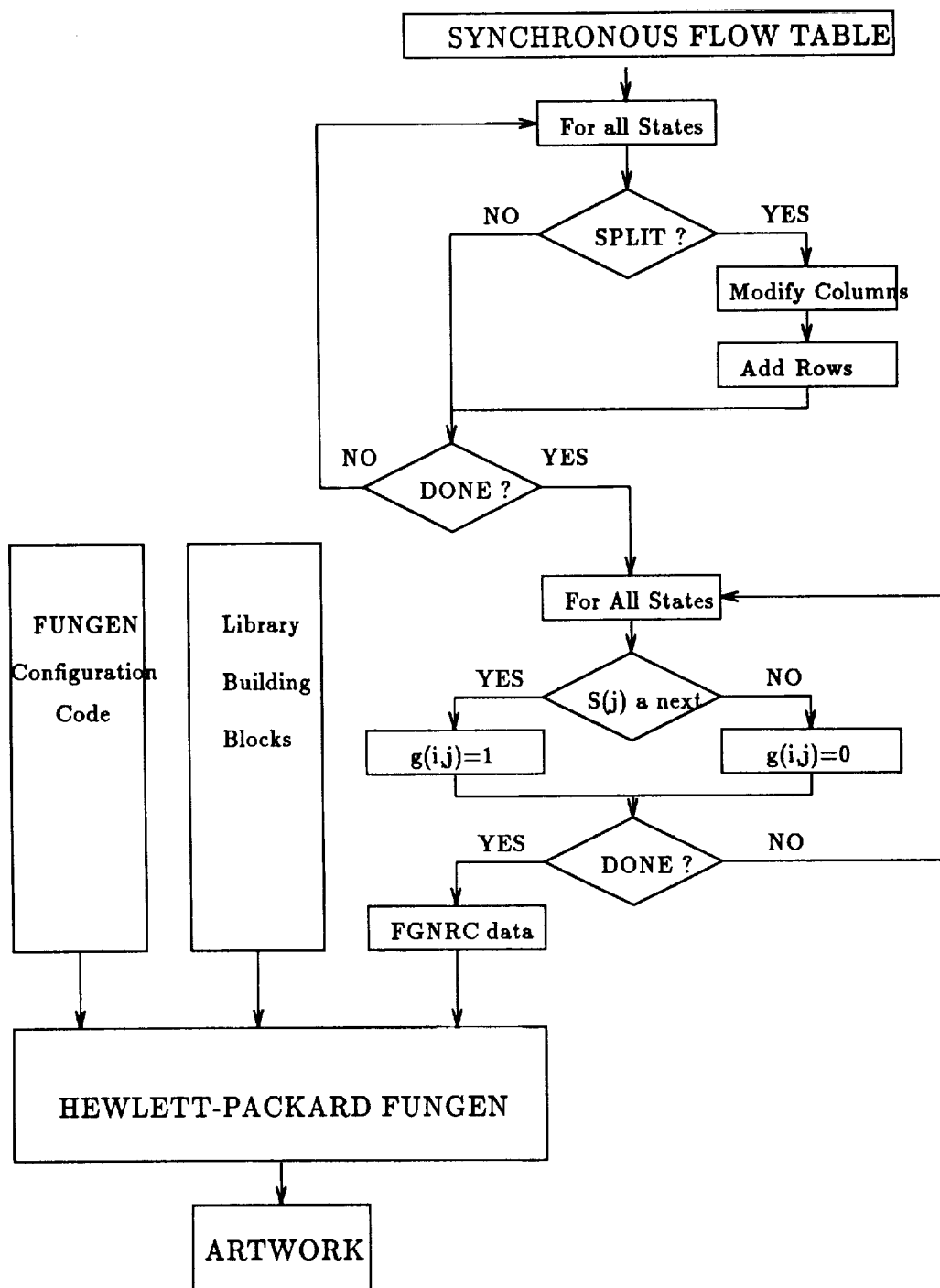


Figure 5: Block diagram of the automatic ILA design system

## 8 Summary

This paper presents an ILA architecture for synchronous sequential circuits. The design procedure is also proposed to realize synchronous sequential ILA circuits by programming the placement of two basic cells, a 2 to 1 multiplexer or a cell of metal wires. The interconnections between ILA cells is only a single route line in both the X and Y dimension. The simplicity and programmability of the procedure significantly reduce the effort in all stages of synchronous sequential circuit implementation, from logic design, circuit design, physical layout to verification.

The ILA design procedure utilizes matrices expression to represent design equations. One of the advantages of using matrices is that they directly indicate the placement of the ILA cells in the realization. An ILA design tool for synchronous sequential circuits has been implemented into a computer system which automatically generates layout artwork from a synchronous sequential machine specification.

## References

- [1] C. Roth, *Fundamentals of Logic Design*, 3rd Ed., St. Paul, Minn., West Publishing, 1985.
- [2] D. Givone, *Introduction to Switching Circuit Theory*, McGraw-Hill, Inc., 1970.
- [3] S. Whitaker, "Design of Asynchronous Sequential Circuits Using Pass transistors," Ph.D Dissertation, University of Idaho, Feb. 1988.
- [4] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp. 238-242.
- [5] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78.
- [6] W. W. Stiehl, "A Mathematical Basis for the Optimal Synthesis of Finite State Machines." Master of Science Thesis, University of Idaho, Moscow, Idaho, June, 1986.

# Single Phase Dynamic CMOS PLA Using Charge Sharing Technique

Y.B Dhong and C.P Tsang

The Department of Computer Science  
The University of Western Australia  
Nedlands, WA 6009, Australia

**Abstract:** This paper presents single phase dynamic CMOS *NOR-NOR* PLA using triggered decoders and charge sharing techniques for high speed and low power. By using the triggered decoder technique, the ground switches are eliminated, thereby making this new design much faster and lower power dissipation than conventional PLAs. By using the charge-sharing technique in a dynamic CMOS NOR structure, a cascading AND gate can be implemented. The proposed PLAs are presented with a delay-time of 15.95 and 18.05 nsec, respectively, which compare with a conventional single phase PLA with 35.5 nsec delay-time. For a typical example of PLA like the Signetics 82S100 with 16 inputs, 48 input minterms ( $m$ ) and 8 output minterms ( $n$ ), the 2-SOP PLA using the triggered 2-bit decoder is 2.23 times faster and has 2.1 times less power dissipation than the conventional PLA. These results are simulated using maximum drain current of 600  $\mu\text{A}$ , gate length of 2.0  $\mu\text{m}$ ,  $V_{DD}$  of 5 V, the capacitance of an input minterm of 1500 fF, and the capacitance of an output minterm of 1500 fF.

## 1 Introduction

CMOS technology has become a vital technology for VLSI because of its high density and low power dissipation. However, it suffers from low speed due to its inherent parasitic capacitance. Thus high-speed CMOS techniques have been vigorously researched. With high speed CMOS it would be possible to implement real-time digital signal-processing applications. As a result, an advanced CMOS technology is expected to remain at the forefront of VLSI technology for many years to come. The problem in designing VLSI systems is of enormous complexity. This problem can be partially simplified by using the more general design principle of a PLA which provides a regular structure. PLAs are also attractive to the VLSI designer because their structure requires a minimum number of separate cell designs, and allows for ease in testing while offering the opportunity for simple, rapid expandability [2].

The delay-time of a dynamic CMOS NOR gate increases very slowly with increasing number of inputs [7]. By cascading two stages of multi-input NOR gates, any desired Boolean function of the input variables can be generated. A CMOS dynamic PLA makes use of these two properties. In CMOS technology the use of PLAs has continued, mainly for regularity of layout and ease of code modification [6]. PLA-based  $n\text{MOS}$  processors

are superior in power dissipation to random-logic based machines by a significant margin, although they are slightly larger in area [1]. Dynamic CMOS PLAs are often used to generate state vectors for lower power microprocessors rather than *n*MOS PLAs. However, the use of ground switches and multiple phases reduces the speed of PLA using conventional dynamic CMOS technology [7,8].

The fastest conventional single phase dynamic CMOS *NOT-NOR-NOR-NOT* PLA has large noise spikes at the floating nodes [3,4]. Hence, a four phase dynamic CMOS *NOT-NOR-NOR-NOT* PLA [8] is commonly adopted. The problem of this four phase scheme is that complex clocks must be generated to drive the dynamic logic circuits. This also reduces speed and requires larger interconnection area. A low-power *NAND-NOT-NOR-NOT* PLA using a simplified addressing scheme is proposed with 244 ns per instruction for a signal processor in paper [4].

In this paper, a family of PLAs using triggered decoders and charge sharing techniques is proposed. They are single phase dynamic CMOS *NOT-NOR-NOR-NOT* PLAs in a sum of products (SOP), called SOP PLA, using 1-bit and 2-bit triggered decoders, respectively. By using the charge sharing technique for the implementation of cascaded AND array in NOR structures, and the triggered input technique for the deletion of ground switches, these PLAs are faster and require lower power dissipation than the conventional single phase dynamic CMOS *NOT-NOR-NOR-NOT* PLA. By using triggered 2-bit decoders on the input during the precharge time, the capacitances of an input minterm of a PLA can be minimized to reduce power [5,6]. Therefore, it is possible to make a faster PLA employing the triggered input decoder circuits and the charge sharing technique.

## 2 Dynamic Single Phase CMOS PLA

CMOS PLA operations may be divided into two classes: pseudo-*n*MOS and dynamic CMOS. Advantages of the pseudo-*n*MOS PLAs include simplicity and small area. Disadvantages are due to the static power dissipation. The dynamic CMOS PLAs generate less power and ground noise than the pseudo-*n*MOS PLAs. The pseudo-*n*MOS PLAs are faster than dynamic CMOS PLAs, but for large PLA layouts the power dissipation is excessive, thus forcing the designer to go to dynamic CMOS.

A modified schematic of a conventional single phase dynamic CMOS *NOT-NOR-NOR-NOT* PLA [3] is shown in Figure 1. This PLA is known to be the fastest [4], but at the expense of the cost of wasted power. The ground switches are charged and discharged every cycle. They connect the sources of all the AND array input transistors, and for layout compactness are built in diffusion. This results in a high capacitance of the order of tens of picofarads. For larger minterms, the additional capacitance of the ground switches is significant.

### 2.1 Triggered Input Logic for Dynamic CMOS Logic

Dynamic CMOS circuits have higher speed and lower chip area than the conventional



static CMOS logic. However, it can only implement noninverting functions. This is because every Domino CMOS gate has to be followed by an inverter and the output of any dynamic CMOS gate cannot be fed directly to another gate. It is possible to avoid this problem by using a triggered 1-bit decoder.

Let us assume that every dynamic triggered 1-bit signal  $(a, a')$  is reset to  $(0,0)$  during the precharge period. A triggered 1-bit decoder signal as a two-valued "one" signal  $A$  is represented as  $(1,0)$ , and the corresponding two-valued "zero" signal  $A$  as  $(0,1)$  during the evaluate period. For example, a two-valued static signal  $A$  can be represented as a triggered 1-bit decoder signal  $(a, a')$  using a two-variable two-valued signal with three states, as shown in Table 1.

Figure 2 shows a triggered 1-bit decoder for dynamic CMOS logic circuits. It consists essentially of a general signal  $a$  and its complementary signal  $a'$ . During the precharge time ( $\bar{\phi} = 1$ ), n-devices (2,6) are on to be precharged to low at the outputs ( $a$  and  $a'$ ), and p-devices (1,5) are off, so the output signals are low, and p-device (3) and n-device (4) together run as an inverter. During the evaluate time ( $\bar{\phi} = 0$ ), when n-devices (2,6) are off and p-devices (1,5) are on, the values of drains of p-devices (1,5) can be transferred at the outputs  $a$  and  $a'$ . Only the logic values of drains in p-devices (1,5) can be transferred to the outputs respectively, because the outputs  $a$  and  $a'$  are low initially during the evaluate time ( $\bar{\phi} = 0$ ). In order to make the same rise time of both a complementary signal  $a'$  and a general signal  $a$  of any input signal, the channel width of p-device 3 must be designed to be larger than that of n-device 4. This decreases the delay-time of input signals at the beginning of the evaluation time.

Figure 3 shows a triggered 2-bit decoder for input signals of a dynamic CMOS PLA. A 2-bit decoder decodes a 2-bit number into 4 output signals. In our case, during precharge time the decoder output signals are set to all "one" or all "zero" depending on the SOP or POS types, respectively. The n-device (14) and the p-device (6) together run as  $\bar{A}B$  gate. This saves power dissipation and improves speed in dynamic CMOS PLA. This is because the number of input nodes is reduced by half leading to reduction of the capacitance of input nodes. If the node  $N_1$  is high, the node  $N_2$  is low, the node  $N_3$  of the drain of p-device (4) is low, and the node  $N_4$  of the gate of n-device (4) is low during precharge time, then the node  $N_1$  becomes low with threshold voltage and slow speed. However, because the node  $N_2$  is low during the precharge time, the transfer of logic "1" only occurs through p-devices (4,5) to the node  $N_2$  with high speed during the evaluation time. During the precharge time the n-devices (13,15,17,19) work as the ground switches in the front array part of PLAs, because they set all input signals of the front array part to "zero". In the case of SOP PLA (see next section), all input signals in the front array part are set to "one" during the precharge time. In this way, these triggered decoder circuits allow the input of static CMOS signals in a dynamic CMOS PLA.

## 2.2 Design of a Single Phase Dynamic CMOS SOP PLA

The schematic of a single phase dynamic CMOS *NOT-NOR-NOR-NOT* sum of product (SOP) PLA using triggered 1-bit decoders is shown in Figure 4. This SOP PLA consists

of triggered 1-bit decoders, the AND array, buffers, and the OR array.

The triggered 1-bit decoder consists of inverters (such as 1), the p-devices loads (such as 4,6) as the ground switches in the AND (front part) array, and functional n-device switches (such as 3,5). The n-device (3) acts as the switch for a general signal and the n-device (5) acts as the switch for a complementary signal.

The AND array consists of loads (such as p-device 11,12), switches (such as n-device 15,16) as the ground switches in the OR (next part) array, and functional switches (such as n-device 19,20) with no ground switch. A conventional dynamic CMOS logic system in two-valued logic using two-variable two-valued logic must use ground switches to prevent a discharge path during precharge time. However, by using triggered 1-bit decoder logic, ground switches are not needed. Thus this reduces power dissipation and improves speed through the omission of the ground switch. Furthermore, the triggered 1-bit decoders set all input signals in the AND array to high and all input signals in the OR array to low during precharge time. In this way, the triggered decoder concept is suitable for a dynamic CMOS PLA system.

Charge sharing is usually a problem in the design of dynamic CMOS AND gates. However, the proposed NOR gates which use charge sharing techniques are suitable for the implementation of the AND array. This charge sharing technique in the AND array overcomes the difficulty of cascading single phase dynamic CMOS gates without the ground switches. All inputs are assumed stable before the evaluation time. During precharge time, when the loads (such as 11,12) are precharged, the input load nodes (such as  $N_1$ ) are charged to high, and all of minterm nodes (such as  $N_2$ ) in AND array are discharged to low because all triggered 1-bit decoder signals are high. When the clock goes high for the evaluation, all loads of input minterms are turned off and the minterm switches n-device (such as 15,16) are turned on. Evaluation paths will exist through the AND array input devices according to the state of the inputs. During evaluation time, the output of the AND array will conditionally charge to high if only all inputs in the minterm of the AND array are high. This keeps all minterm lines to virtual ground except the selected ones, which have all input transistors connected to them turned off with all existing charges remaining shared. These AND gates work as *NOT-NOR* gates. Also, a minimum capacitance ratio value of  $C_{N_1}/C_{N_2}$  of 2:1 is required, where  $C_{N_1}$  is the capacitance of the node  $N_1$  and  $C_{N_2}$  is the capacitance of the node  $N_2$ .

The speed of the AND array depends on the minterm switches (such as 15,16), and their maximum drain current depends on the n-device (such as 19,20) and the n-device (such as 15,16). Thus in the range of the maximum drain current ( $600 \mu\text{A}$ ) we can increase slightly more the width of the n-devices (15,19) to  $6.5 \mu\text{m}$ , to improve speed over that of a conventional CMOS PLA of  $4 \mu\text{m}$ . Figure 5 shows the simulated waveforms of internal nodes. If A and B are low,  $V_4$  is  $\phi$  signal as the input trigger voltage,  $V_{12}$  is the voltage of NOR gate using charge sharing at the selected node  $N_3$ , and  $V_{13}$  is the pumping voltage at the unselected node  $N_2$ .

Figure 6 shows the simulated waveforms of selected internal nodes.  $V_2$  is the selected output voltage at  $O'$ ,  $V_4$  is  $\phi$  signal as the input trigger voltage,  $V_{12}$  is the voltage at the

selected AND array node  $N_3$  and its steady state voltage is the same as  $C_{N_1}V_{DD}/(C_{N_1} + C_{N_2})$ , and  $V_{16}$  is the voltage at the selected OR array node  $N_5$  if A and B are low.

Figure 7 shows the relationship between the pumping voltage and the width ratio. This PLA was simulated using SPICE3d1 based on the channel width of the n-device 15 ( $W_{15}$ ) at  $6.5 \mu m$ , and that of the n-device (19) ( $W_{19}$ ) at  $6.5 \mu m$ . Although the channel width ratio  $w$  increases linearly with speed, it demands the increase of the pumping voltage with a decrease in the noise margin. Thus in order to prevent the incorrect operation resulted from pumping phenomena (i.e., a reduction of the noise margin) at the nodes (such as  $N_2$ ) during the evaluation time, and to keep the virtual ground, an optimal channel width ratio  $W_{15}/W_{19}$  of 1:1 can be used. Therefore, to make a minimum pumping voltage, the optimal width ratio  $w$  has "one" and in the worst case only, one input of the multi-input AND gate is high.

Figure 8 shows a dynamic CMOS buffer using a one-way NOR gate.. The width of the n-device (1) must be designed to be shorter ( $5 \mu m$ ) to prevent the discharge by the pumping voltage relative to the ground switch ( $15 \mu m$ ). In buffers, the first NOR buffers (such as 35,36) should be designed so that the logic threshold value of the NOR buffer is a lower value than  $V_{DD}/2$ . This measure improves the speed. These buffers can be used to improve speed because the node  $N_2$  has larger capacitance due to many input variables. The rising time of an input minterm depends predominantly on the resistance of the n-device (15). Some delay will be incurred due to the finite pull-up time. The inverters (43,44) are used for synchronizing the load signal with the triggered decoder signals in input minterms. This AND array using the charge sharing technique does not require the input tracking lines in the SOP PLA.

The OR array consists of loads (such as p-device 31,32), inverters (such as 33,34), and functional switches (such as n-device 27,28) with no ground switch. Charge is dissipated only in the selected output lines themselves. The power dissipation in the OR array is minor compared to the AND array.

By using triggered 2-bit decoders on the input during the precharge time, a number of input minterms of a PLA can be minimized to reduce power and to improve speed. Therefore, it is possible to make a faster PLA with no ground switch. This SOP PLA is suitable for the implementation of the dynamic CMOS PLA which has a lower number of the AND array minterms and a greater number of the OR array minterms.

### 3 Simulation Results and Conclusions

To compare the performance of the single phase dynamic CMOS PLAs, each of the PLAs described in previous sections was simulated using SPICE3d1. The simulated waveforms of the various single phase dynamic CMOS PLA are shown in Figure 9 and 10. The input waveforms are V(35) and V(4), and the output waveforms of the front array are V(26) and V(16). the output waveforms are V(3) and V(2) in Figure 9 and Figure 10, respectively. In simulation, the drain maximum current of  $600 \mu A$  in the input functional n-device is used, the gate length is  $2.0 \mu m$ , the  $V_{TO}$  of n-device is  $0.71 V$ , the  $V_{TO}$  of

p-device is 0.80 V, the drain capacitance  $C_D$  in the n-device is assumed 11.43 fF, the gate capacitance  $C_G$  in the n-device is assumed 2.9 fF,  $V_{DD}$  is 5 V, the node number of an input minterm is 130 with 1.5 pF capacitances, and the node number of an output minterm is 130 with 1.5 pF capacitances.

Table 2 shows a comparison of simulation results for various single phase CMOS PLA types, where both the input minterm and the output minterm are assumed to have the same capacitance, the number of input minterms is  $m$  and the number of output minterms is  $n$ .  $T_f$  is the delay-time of the front array of a PLA and  $P_f$  is the normalized average power of the front array of a PLA.  $T_b$  is the delay-time of the back array of a PLA and  $P_b$  is the normalized average power of the back array of a PLA.  $T_t$  is the total delay-time of a PLA and  $P_t$  is the total normalized average power of a PLA. The worst case total delay time of a conventional single phase dynamic CMOS PLA is 35.5 ns. The SOP PLA using the triggered 1-bit decoder and the 2-SOP PLA using the triggered 2-bit decoder are 2 and 2.23 times faster, respectively, than the conventional CMOS PLA.

The normalized average power can be considered as the total charges in a minterm. The front array in the conventional PLA has the number of average selected minterms of  $\frac{m}{2} + 1$ , where the "1" is the input tracking line. The back array has the number of average selected minterms of  $\frac{n}{2}$ . The "5" is the charge of a minterm and the "4" is the charge of a ground switch. The proposed AND array using charge sharing technique in the SOP PLA has the number of average selected minterms of  $\frac{m}{2}$  and  $\frac{n}{2}$ , respectively. The selected minterm has wasted  $\frac{10}{3}$  normalized charge and the unselected minterm has wasted 10 normalized charge. The charge of the front array in the 2-SOP PLA is a half charge of that in the SOP PLA because of using the triggered 2-bit decoders. Thus the proposed PLA structures are faster and require lower power dissipation than the conventional single phase dynamic CMOS *NOT-NOR-NOR-NOT* PLA, because of the elimination of the ground switch. For a typical example of PLA like the Signetics 82S100 with 16 inputs, 48 input minterms ( $m$ ) and 8 output minterms ( $n$ ), the 2-SOP PLA using the triggered 2-bit decoder is 2.23 times faster and has 2.1 times less power dissipation than the conventional PLA. Therefore, the proposed 2-SOP PLA using the triggered 2-bit decoder is a faster dynamic CMOS PLA, and this PLA has no input tracking line.

## References

- [1] P. Cook, C.W. Ho, and S. Schuster. A study in the use of PLA based macros. *IEEE Journal Solid-State Circuits*, SC-14:833-840, October 1979.
- [2] Eugene D. Fabricius. *Introduction to VLSI Design*. McGraw-Hill in Electrical Engineering: New York, 1990.
- [3] D. Hodges and H. Jackson. *Analysis and Design of Digital Integrated Circuits*. McGraw-Hill: New York, 1983.
- [4] Alfredo R. Linz. A low-power PLA for a signal processor. *IEEE Journal of Solid-State Circuits*, 26(2):107-115, February 1991.

- [5] F.J. Pelayo, A. Prieto, A.Lloris, and J. Ortega. CMOS current-mode multivalued PLA's. *IEEE Trans. on Circuits and Systems*, 38(4):434-441, April 1991.
- [6] Tsutomu Sasao. On the optimal design of multiple-valued PLA's. *IEEE Trans. on Comp.*, 38(4):582-592, April 1989.
- [7] Masakazu Shoji. *CMOS Digital Circuit Technology*. Computer System. Prentice Hall, Englewood Cliffs, New Jersey, Computing Science Research Center AT&T Bell Laboratories, 1988.
- [8] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design A Systems Perspective*. The VLSI Systems Series. Addison-Wesley, Massachusetts, 1985.

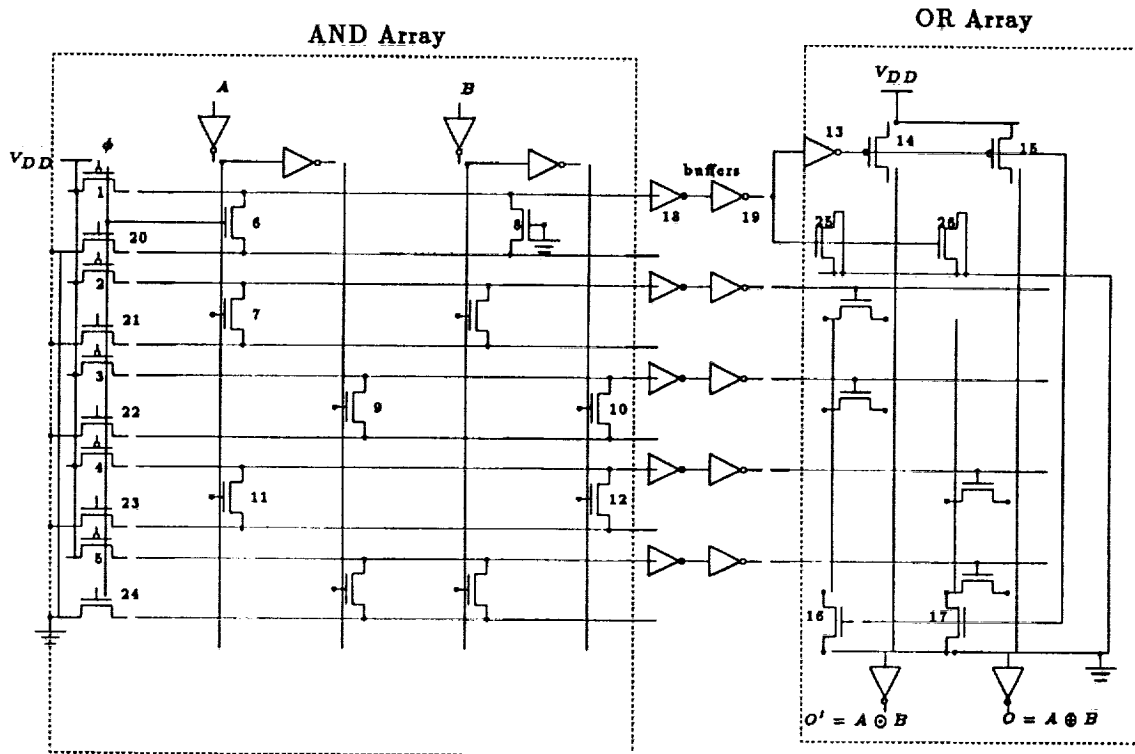


Figure 1: Conventional single phase (NOT-NOR)-(NOT-NOT)-(NOR-NOT) PLA

Time	1-bit signal	triggered 1-bit decoder signal			
	$A$	$a$	$a'$	$\bar{a}$	$\bar{a}'$
precharge	*	0	0	1	1
evaluate	0	0	1	1	0
	1	1	0	0	1

Table 1: Encoding a 1-bit signal into a triggered 1-bit decoder signal using a two-variable two-valued signal in a dynamic CMOS logic system

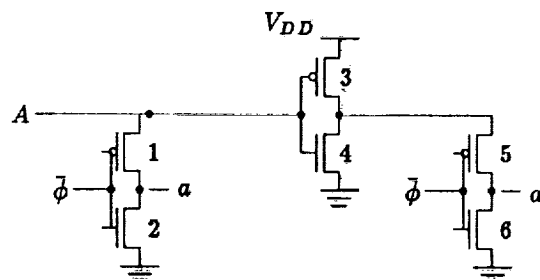


Figure 2: CMOS implementation of a triggered 1-bit decoder for dynamic CMOS logic circuits

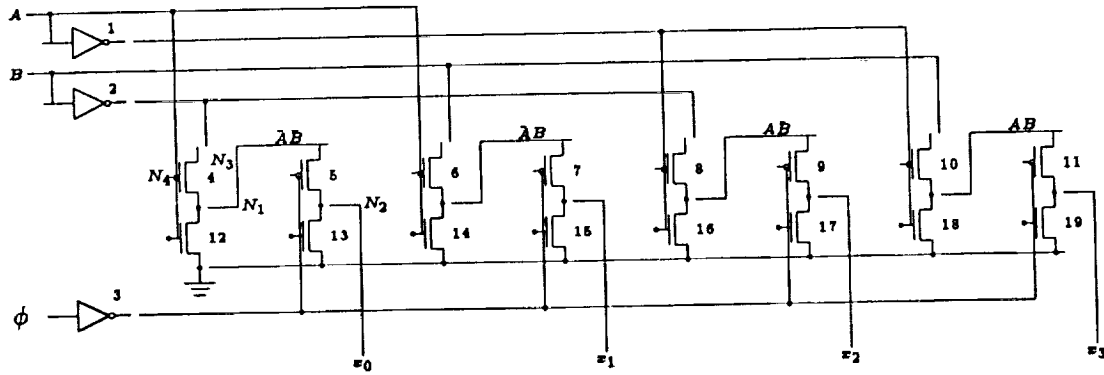


Figure 3: Static CMOS implementation of a triggered 2-bit decoder

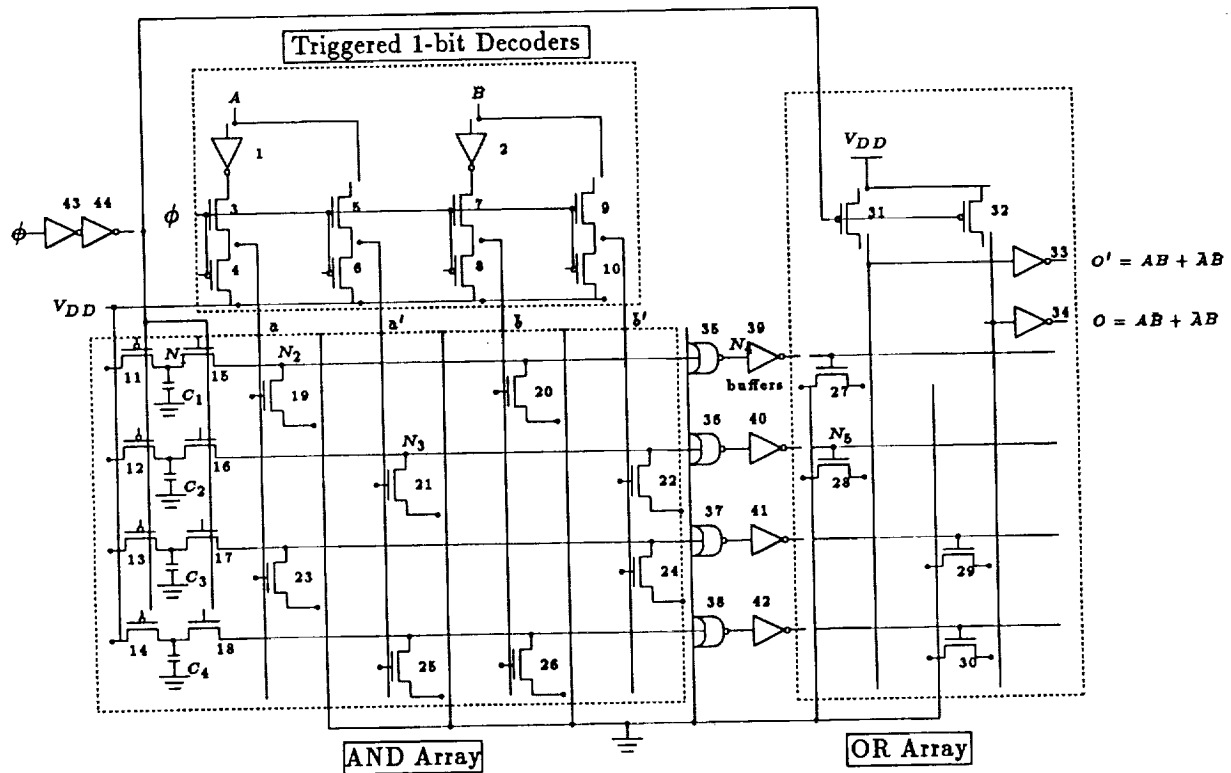


Figure 4: Single phase dynamic CMOS (NOT-NOR)-(NOT-NOT)-(NOR-NOT) PLA in a sum of products (SOP) using triggered 1-bit decoders

Techniques	$T_f(ns)$	$P_f$	$T_b(ns)$	$P_b$	$T_t(ns)$	$P_t$
Conventional	17.75	$(\frac{m}{2} + 1)5 + (m + 1)4.3$	17.75	$\frac{n}{2} \cdot 5 + 4.3n$	35.5	$6.8m + 6.8n + 9.3$
SOP	10.55	$\frac{m}{2}10 + \frac{m}{2} \cdot \frac{10}{3}$	7.5	$\frac{n}{2} \cdot 5$	18.05	$6.7m + 2.5n$
2-SOP	8.45	$\frac{m}{2}5 + \frac{m}{2} \cdot \frac{5}{3}$	7.5	$\frac{n}{2} \cdot 5$	15.95	$3.3m + 2.5n$

Table 2: Comparison of simulation results for various single phase CMOS PLA types

## 2.2.10

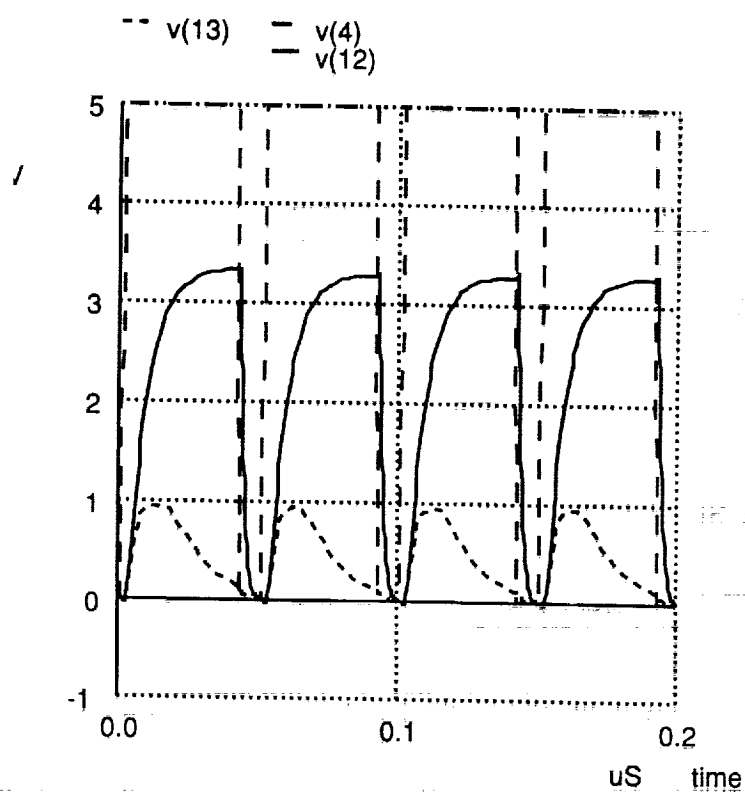


Figure 5: Simulated waveforms of internal nodes

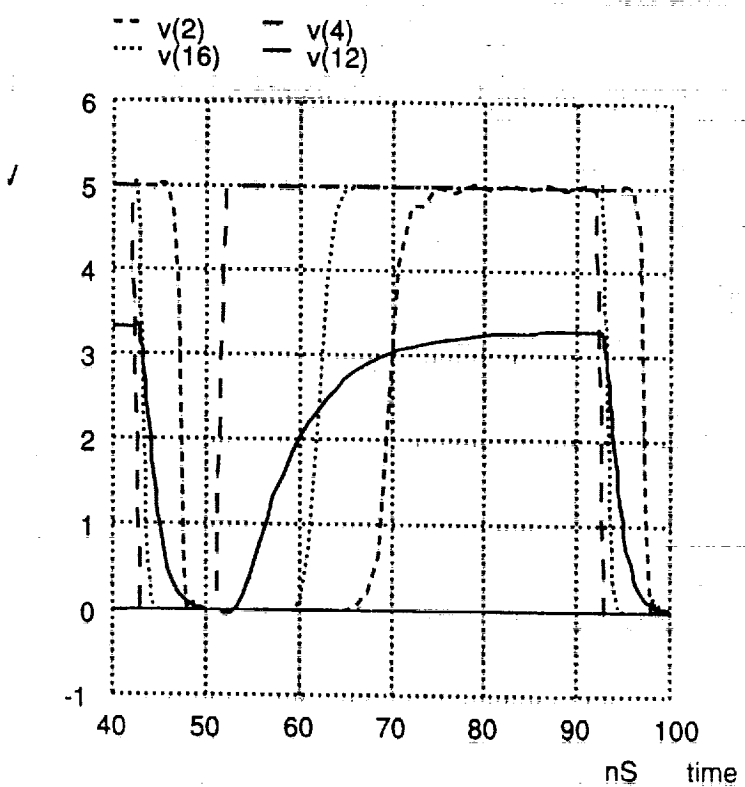


Figure 6: Simulated waveforms of internal selected nodes in a single phase dynamic CMOS SOP PLA using the triggered 1-bit decoder



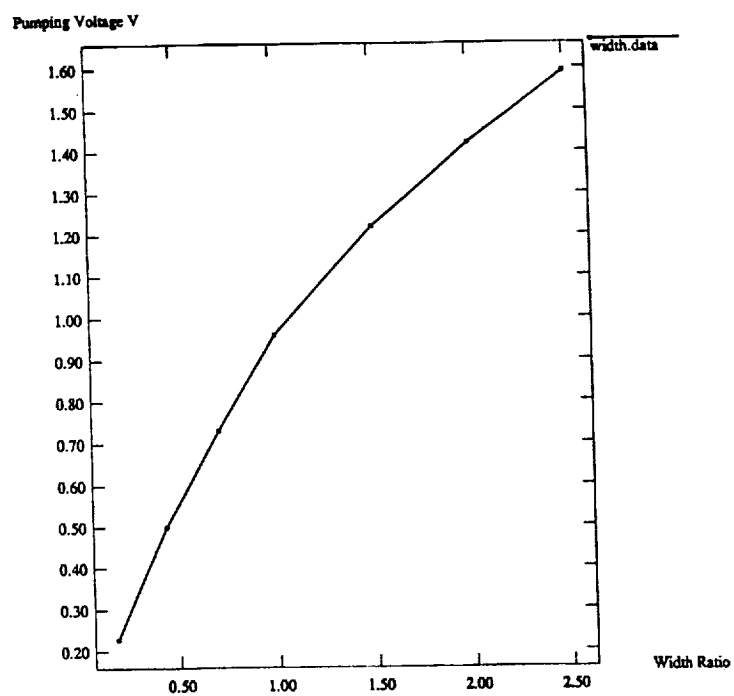


Figure 7: Simulated pumping voltage versus channel width ratio  $w = W_p/W_n$

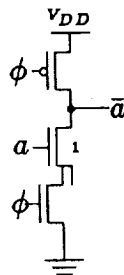


Figure 8: Dynamic CMOS buffer using an one-way NOR gate

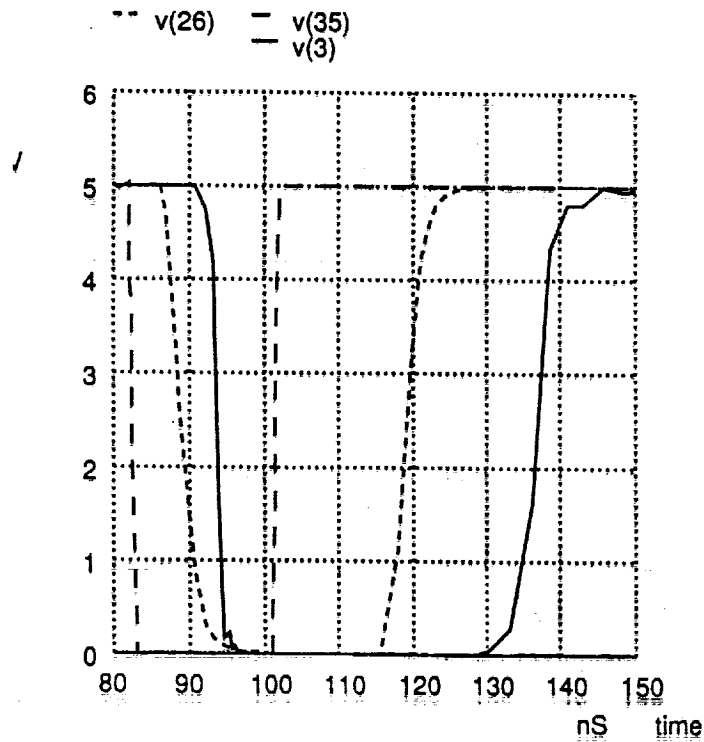


Figure 9: Simulated waveforms of a conventional single phase dynamic CMOS PLA.  $V_{35}$  is the triggered voltage of  $\phi$ ,  $V_3$  is the output voltage at the node  $O'$ , and  $V_{26}$  is the output voltage of the AND array at the gate of p-device (14)

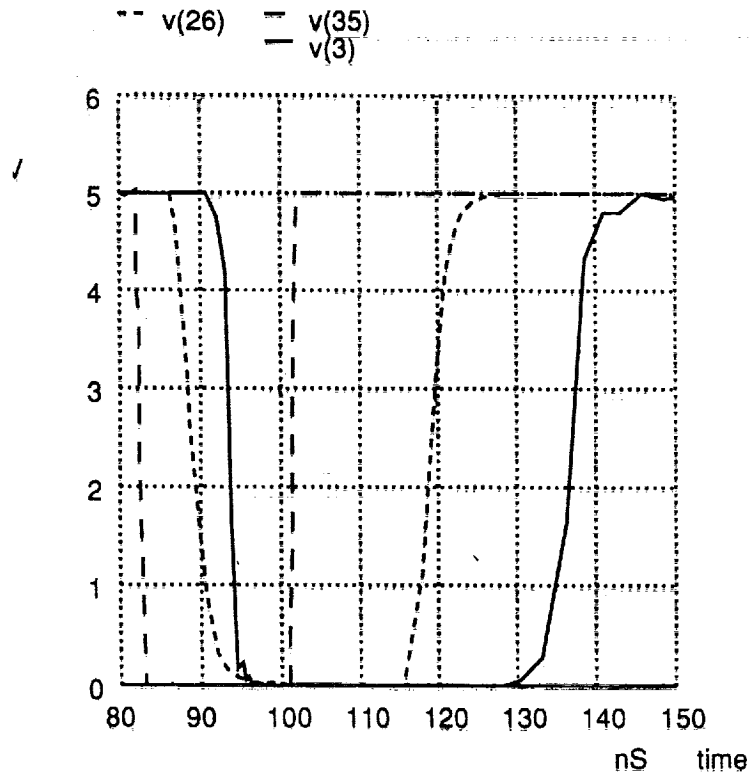


Figure 10: Simulated waveforms of a single phase dynamic CMOS SOP PLA using the triggered 2-bit decoder. All voltage numbers are the same as SOP PLA using the triggered 1-bit decoder.

# An SEU Immune Logic Family

J. Canaris

NASA Space Engineering Research Center for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract** - A new logic family, which is immune to single event upsets, is described. Members of the logic family are capable of recovery, regardless of the shape of the upsetting event. Glitch propagation from an upset node is also blocked. Logic diagrams for an Inverter, Nor, Nand and Complex Gates are provided. The logic family can be implemented in a standard, commercial CMOS process with no additional masks. DC, transient, static power, upset recovery and layout characteristics of the new family, based on a commercial  $1\mu\text{m}$  CMOS N-Well process, are described.

## 1 Introduction

Historically, the emphasis on Single Event Upset (SEU) research has been devoted to memory circuits [1]-[16]. Memory circuits perform vital functions in any digital system, as program stores, temporary registers and as elements of state machines which control digital circuits. An SEU, or soft error, caused by a charged particle striking a diffusion region in a memory element can prove catastrophic to an electro-mechanical system which relies upon that memory element for communication or control. Great effort has been made to find memory structures which are immune to SEUs, or at least mitigate the effects of an upsetting event. The design of SEU immune memories, whether RAM or Flip-Flops, has tended to ignore system level problems, such as an SEU of a combinational logic gate which is sampled by a memory circuit, or an upset of a control signal such as a clock line or mux select. It has been shown [17,18] that transients propagated out of or into memory elements is indeed a real problem. Research, to find general logic gate structures which are SEU immune, has been primarily limited to resistive or capacitive hardening, which are basically low pass filtering approaches [17,19,20]. Kang and Chu [21] present a logic/circuit design approach but the CMOS inverter buffers are susceptible to particle hits on the p-type diffusion. The pre-charged output node is susceptible to a particle strike on the n-type diffusion if the pulldown chain does not evaluate low. More recently [16] and [22] have presented memory cells based on logic/circuit design techniques. Only [22] addresses the issue of glitch propagation.

This paper presents a complete logic family which is SEU immune. Members of the family are constructed, using logic/circuit design techniques, to recover from an SEU, regardless of the shape of the upsetting event. It is also shown that the logic family can prevent glitch propagation from an upset node. The logic family can be implemented in a standard, commercial CMOS process without any additional processing steps. The DC,

transient, static power, upset recovery and layout characteristics of the new family, based on a commercial  $1\mu\text{m}$  CMOS N-Well process, are presented in this paper.

Section 2 provides circuit configurations of members of the logic family, including an Inverter, 2-input Nand, 2-input Nor, 3-input OrNand and a 3-input AndNor. In addition, a description of the SEU recovery mechanism is presented and a means for extending this mechanism to logic structures in general is provided. DC characteristics of the SEU immune inverter are described in Section 3. Noise margins, gain characteristics and the effects of device ratioing and threshold voltages are discussed. Section 4 provides simulation results which show that the SEU recovery mechanism is independent of the duration or shape of the upsetting event. Blocking of glitch propagation is also presented. Section 5 presents circuit switching speed results based on pair-delay simulations. The effects of device ratioing on switching speed are also discussed. Static power considerations are presented in Section 6 and physical layout issues are presented in Section 7. Section 8 provides a summary and conclusions.

## 2 An SEU Immune Logic Family

The literature related to SEU immune combinational logic is sparse and has provided few clues as to what would be necessary to design a logic family which provides immunity to single event upsets. Whitaker has, however, provided a concise summary of fundamental concepts which can be used in the design of SEU immune memory circuits [22]. First, information must be stored in two different places. This provides a redundancy and maintains a source of uncorrupted data after an SEU. Second, feedback from the noncorrupted location of stored data must cause the lost data to recover after a particle strike. Finally, current induced by a particle hit flows from the n-type diffusion to the p-type diffusion. If a single type of transistor is used to create a memory cell then p-transistors storing a 1 cannot be upset and n-transistors storing a 0 cannot be upset. An understanding of these three concepts and close examination of the memory circuit presented in [22] has provided the key to the design of an SEU immune logic family.

Figure 1 is a transistor level logic diagram of an SEU immune inverter. The inverter consists of two transistor networks, a p-channel network and an n-channel network. All devices are enhancement mode transistors. The inverter is a two input/two output logic device with  $P_{in}$  driving only p-channel devices and  $N_{in}$  driving only n-channel devices. Node  $P_{out}$  can provide a source of 1's which cannot be upset and node  $N_{out}$  provides a source of 0's which cannot be upset. Transistor M2 is sized to be weak compared to M1 and transistor M3 is sized to be weak compared to M4. The SEU recovery mechanism works as follows. When the inputs to the inverter are 0,  $P_{out}$  and  $N_{out}$  are at a 1. In this state only  $N_{out}$  can be corrupted by an upset. If  $N_{out}$  is hit, driving the node to a 0, transistor M2 will turn on but cannot overdrive M1.  $P_{out}$  will remain at a 1, transistor M3 will remain on, pulling  $N_{out}$  back up to a 1. Conversely, if  $P_{in}$  and  $N_{in}$  are 1,  $P_{out}$  and  $N_{out}$  will be at 0 and only  $P_{out}$  can be upset. If  $P_{out}$  is hit, driving the node to a 1, transistor M3 will turn on but being weak compared to M4,  $N_{out}$  will remain pulled down to a 0.

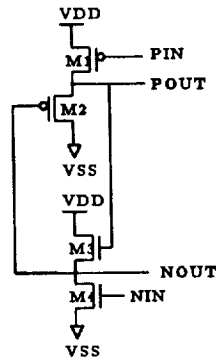


Figure 1: SEU Immune inverter.

The inverter follows the fundamental principles for SEU immunity and is therefore made SEU immune.

It is readily apparent that the inverter design concepts can be applied to any logic gate to provide SEU immunity. Figures 2,3,4 and 5 are the transistor level logic diagrams of a two-input NAND, two-input NOR, three-input OrNand and three-input AndNor respectively. In general, an SEU immune logic gate, implemented with this technique, requires  $2n + 2$  transistors,  $n$  being the number of gate inputs. In comparison, classical CMOS requires  $2n$  transistors to implement a gate.

The logic family described here can provide transient suppression of an upset event as well as recovery from the upset. Networks of logic gates are connected such that  $P_{out}$  only drives p-channel devices and  $N_{out}$  only drives n-channel devices. If  $P_{out}$  is upset, driving the node to a 1, the p-transistor being driven will be turned off momentarily without affecting the output of the following stage. If  $N_{out}$  is upset, driving the node to a 0, the n-transistor being driven will be turned off momentarily without affecting the output of the following stage.

The above description obviously overlooks some of the circuit design issues which would be faced by someone wishing to design with this logic family. The family, although implemented in a CMOS process is ratioed logic, with a ratioing occurring between transistors M1 and M2 and between transistors M3 and M4. This logic family, therefore, bears a closer resemblance to NMOS than it does to CMOS. Additionally, threshold voltages become a design issue because of the enhancement mode transistors being used to pull up  $N_{out}$  and to pull down  $P_{out}$ . Design implementation issues related to ratioing and threshold voltages are presented in the following sections.

### 3 Inverter DC Characteristics

The DC transfer function,  $\frac{V_{out}}{V_{in}}$ , of an inverter provides several useful pieces of information about a logic family. Noise margin, inverter gain and inverter switch points are all characteristics which can be determined from a plot of  $V_{out}$  versus  $V_{in}$ . A DC transfer function plot can also show if hysteresis is present. The SPICE [23] circuit simulator was

NAME	PARAMETER SET	VOLTAGE RANGE	TEMP.
WCLVHT	SLOW N SLOW P	4.1V	140°C
WCHVHT	SLOW N SLOW P	5.5V	140°C
WCLVLT	SLOW N SLOW P	4.1V	-55°C
WCHVLT	SLOW N SLOW P	5.5V	-55°C
BCLVHT	FAST N FAST P	4.1V	140°C
BCHVHT	FAST N FAST P	5.5V	140°C
BCLVLT	FAST N FAST P	4.1V	-55°C
BCHVLT	FAST N FAST P	5.5V	-55°C
FNSPLVHT	FAST N SLOW P	4.1V	140°C
FNSPHVHT	FAST N SLOW P	5.5V	140°C
FNSPLVLT	FAST N SLOW P	4.1V	-55°C
FNSPHVLT	FAST N SLOW P	5.5V	-55°C
SNFPLVHT	SLOW N FAST P	4.1V	140°C
SNFPHVHT	SLOW N FAST P	5.5V	140°C
SNFPLVLT	SLOW N FAST P	4.1V	-55°C
SNFPHVLT	SLOW N FAST P	5.5V	-55°C

Table 1: DC Transfer Function Simulation Cases

used to generate DC transfer functions for the SEU immune inverter described in Section 2. Results of these simulations will be presented here.

In a classical family of logic, such as NMOS, PMOS or CMOS a transistor  $\beta$  is defined to be the product of the process gain factor,  $K'$ , and the transistor aspect ratio,  $\frac{W}{L}$ . That is  $\beta_{TRAN} = K'(\frac{W}{L})$ . The inverter  $\beta$  is defined as the ratio of the pullup  $\beta$  and the pulldown  $\beta$ , or  $\beta_{INV} = \frac{\beta_{PU}}{\beta_{PD}}$ . The logic family described in Section 2 is a ratioed logic family. In this case the ratioing occurs between the same type devices, and the  $K'$  term cancels in  $\beta_{TRAN}$ . Therefore,  $\beta_{TRAN} = \frac{W}{L}$ . In this case it is more useful to define transistors as strong (M1,M4) and weak (M2,M3), instead of the traditional pullup and pulldown. To complicate matters further,  $\beta_{INV}$  now has two components,  $\beta_N$  and  $\beta_P$  which are not necessarily equal. For the simulations presented here,  $\beta_{INV} = \beta_N = \beta_P = \frac{\beta_{STRONG}}{\beta_{WEAK}}$ .

As weak is a relative term and it was unknown what effect ratioing would have on DC characteristics, simulations were run over 16 process parameter/voltage/temperature cases on 15 values of  $\beta_{INV}$  ranging from  $\frac{1}{8}$  to  $\frac{8}{1}$ . Table 1 lists the 16 simulation cases. It was necessary to run these 16 cases in order to determine what effect processing variations would have on the SEU immune inverter. The temperature and voltage ranges cover those required by military specifications of integrated circuits.

Once the DC simulations were performed, an inverter gain and noise margin analysis was undertaken. It is known that ratioed logic, particularly when threshold voltage effects are involved, has lower noise margins than non-ratioed CMOS logic. Ratioing will also effect the gain of a logic gate. If the gain is too low a signal will die out after only a few logic stages. In the case of the SEU immune inverter, under the WCLVLT case, gains of 1 or

NOISE MARGIN LOW			
LOW	0.20V	$\beta_{INV} = \frac{1}{1}$	BCHVHT
HIGH	0.74V	$\beta_{INV} = \frac{1}{2}$	FNSPHVLT
NOISE MARGIN HIGH			
LOW	0.30V	$\beta_{INV} = \frac{1}{2}$	SNFPLVHT
HIGH	1.07V	$\beta_{INV} = \frac{1}{1}$	WCLVLT
INVERTER GAIN VARIATIONS			
LOW	1.6		
HIGH	11.3		

Table 2: Noise margin and inverter gain variations

less were attained for  $\beta_{INV}$  of  $\frac{1}{8}$  and  $\frac{1}{7}$ . Additionally negative noise margins were attained for  $\beta_{INV}$  of  $\frac{1}{6}$ ,  $\frac{1}{5}$  and  $\frac{1}{4}$ . These  $\beta$ s are of course unusable in a design. Both noise margin low (immunity from positive spikes) and noise margin high (immunity from negative spikes) were analyzed for  $P_{out}$  and  $N_{out}$ . Table 2 provides a summary of this analysis.

The inverter DC simulations eliminated 5  $\beta_{INV}$  from further consideration and showed that several more could prove marginal in a design. With the DC analysis complete, the SEU recovery ability of the inverter could be investigated. The results of this investigation are presented Section 4.

## 4 SEU Recovery Results

To verify the SEU recovery ability and the transient suppression characteristics of the SEU immune inverter, described in Section 2, SPICE simulations were run over the same 16 cases described in Section 3. Both  $P_{out}$  and  $N_{out}$  were tested. Since inverters with  $\beta_{INV} < \frac{1}{3}$  were rejected during DC analysis only 10  $\beta_{INV}$ , ranging from  $\frac{1}{3}$  to  $\frac{8}{1}$ , were simulated at this stage. The SEU immunity of the logic family was shown to be independent of processing parameters, temperature or supply voltage. The error recovery mechanism is provided by the logical feedback of transistors M2 and M3 and the ratioing of transistor strengths. The recovery mechanism is also not dependent upon the wave shape of the current pulse which upsets the node.

The simulation circuit used to test the recovery mechanism consisted of a chain of 3 identical inverters. No parasitic capacitance other than self-capacitance and that seen at the inputs to the next stage was added to the circuit. The inputs to the first inverter were set up to the proper initial conditions. A voltage controlled current source was connected to the node to be upset. This provided a means to inject charge into the node without attaching any parasitic capacitance. Additionally an ideal diode, emulating the parameter dependent source/drain to substrate/well diodes, was attached to the node. This diode did not create any additional capacitance. A current pulse, with a duration of 10ns, and a magnitude sufficient to forward bias the source/drain diode, was applied to the node. The 10ns pulse width was chosen because it was longer than the propagation

delay through the inverter as well as being longer than a real SEU. Recovery from an SEU was shown to be independent of parameter/voltage/temperature cases. Although all  $\beta_{INV}$  cases recovered, SEU recovery time was dependent upon  $\beta_{INV}$ . Faster recovery times were noted for  $\beta_{INV} > \frac{1}{1}$ .

Besides being able to recover from an upset event an SEU immune logic family must be able to suppress the propagation of transients out of the upset node. Due to the P-net driving P-net and N-net driving N-net configuration described in Section 2, the logic family presented in this paper should be able to suppress glitches caused by an SEU. SPICE simulations verified that this is the case. The simulation circuit used to test transient suppression was the same as that used for testing upset recovery. In this case, however, a 1ns current pulse was applied to the upset node. This pulse duration is closer to what one would expect from a real SEU. Transient suppression was measured at the output of the inverter being driven by the upset node. If the magnitude of the glitch on this output was within the noise margin, for the parameter/voltage/temperature case and  $\beta_{INV}$  being simulated, the transient was considered suppressed. Results of these simulations indicated that transient suppression was dependent upon simulation cases as well as  $\beta_{INV}$ . In fact, any  $\beta_{INV} \leq \frac{1}{1}$  was rejected as unusable, in a design, due to poor transient suppression abilities.

The seven ratios with  $\beta_{INV} > 1$  remaining after the SEU recovery/transient suppression simulations were subjected to a transient analysis to determine switching speeds of the SEU immune logic family. These results are presented in Section 5.

## 5 Transient Analysis of the SEU Immune Inverter

With a modern CMOS process it is possible to attain inverter gate delays of 1ns or less. For an SEU immune logic family to be of interest to the VLSI design community the inverter described in Section 2 should have a gate delay at least in the ns range. Transient analysis simulations show that this is possible. SPICE simulations were run over the same 16 cases described in Sections 3 and 4. The simulation circuit was a chain of 7 identical inverters. Each inverter was loaded with a 1000pF linear capacitor. This large capacitor swamped out any voltage dependent capacitors associated with transistor source/drain regions as well as gate capacitances seen by the inverter outputs. The first inverter in the chain was excited by a step function, and pair delay information was extracted from the output. A pair delay is defined to be the delay, measured from mid-point to mid-point of the voltage swing, through a pair of inverters. This delay contains both a time delay rise and a time delay fall. In non-ratioed logic, such as classical CMOS, inverters are designed to have equal rise and fall times. In a ratioed logic family it is not always possible to design for equal rise and fall times, therefore pair delay information is more useful. In this case 4 pair delay values were computed, delay from a rising edge and from a falling edge, for both  $P_{out}$  and  $N_{out}$ . The longest delay of these was chosen as the worst case delay. At the outset it was unknown which parameter/voltage/temperature case would prove to be that of worst case speed. In classical CMOS it would be WCLVHT. For this logic family



Pairdelay Chart ( $C_{load} = 1000pF$ , $Delay = \mu s$ )										
$\beta_{INV}$	FeedBack Transistor Width ( $L = 1.0\mu m$ )									
	$2.4\mu m$	$4.8\mu m$	$7.2\mu m$	$9.6\mu m$	$12.0\mu m$	$14.4\mu m$	$16.8\mu m$	$19.2\mu m$	$21.6\mu m$	$24.0\mu m$
$\frac{2}{1}$	143	54	34	25	19	16	14	12	11	9
$\frac{3}{1}$	107	41	26	19	15	12	10	9	8	8
$\frac{4}{1}$	90	35	22	16	13	11	9	8	7	6
$\frac{5}{1}$	82	32	20	15	12	9	8	7	7	6
$\frac{6}{1}$	75	29	18	13	11	9	8	7	6	5
$\frac{7}{1}$	70	27	17	12	10	8	7	6	5	5
$\frac{8}{1}$	66	25	16	12	9	8	7	6	5	5

Table 3: Pair delay results.

it also proved to be WCLVHT. Simulations were run on all of the surviving  $\beta_{INVS}$ , with ten different transistor widths, ranging from  $2.4\mu m$  to  $24.0\mu m$ . Pair delay charts for each  $\beta_{INV}$  were constructed. A table of pair delay versus transistor width is provided in Table 3. As expected, because delay is inversely proportional to width, pair delays decrease as a function of transistor width. Speed, another useful design measure, is the linear function,  $\frac{1}{delay}$ .

From the results of the SEU recovery ability, described in Section 4, and the pair delay information in this section, it would seem that  $\beta_{INV} = \infty$  would be the best choice. However, as in all engineering endeavors there is a practical limit to the choice of  $\beta_{INV}$ . Both power dissipation and physical layout constraints must be considered. Section 6 and Section 7 will discuss these issues, as they relate to the SEU immune inverter.

## 6 Static Power

In Section 2 it was stated that the SEU logic family presented in this paper was, in some regards, more closely related to NMOS than CMOS. Due to the ratioing between the normal transistors and the feedback transistors, and the effects of threshold voltages, this logic family dissipates static power. SPICE simulations were run, with the same cases described in previous sections, to characterize this power dissipation, and the effects of  $\beta_{INV}$  on it. As expected, power dissipation increased with  $\beta_{INV}$ . The power dissipation was worst under BCHVLT conditions for both input high and input low conditions. Static power consumption may place a limit on the number of SEU immune gates which can be placed on an integrated circuit.

## 7 Physical Layout

The SEU immune logic family presented in this paper can be implemented in a standard CMOS process, using standard layout design rules. The family does, however, have characteristics which makes physical layout of the family different than a classical CMOS layout. A classical inverter, for example, requires a minimum of two lines, the input and the output, crossing the well boundary. The SEU immune inverter has two separate inputs,  $P_{out}$

and  $N_{out}$ , but they need not cross the well boundary. However, there are two feedback lines which must cross. Additionally, both VDD and VSS are required for both n-transistors and p-transistors, whereas a classical inverter only requires one power supply for each transistor type. The signal connections are more complicated in the SEU immune logic family than in classical CMOS. In addition, the SEU immune logic family has two more transistors than does classical CMOS. One should, therefore, expect that layout densities would be less for the SEU immune logic family. As designers acquire more experience with layout considerations the attained densities should improve.

attained.

## 8 Summary and Conclusions

This paper presented a complete logic family which is SEU immune. Members of the family are constructed, using logic/circuit design techniques, to recover from an SEU, regardless of the shape of the upsetting event. It was also shown that the logic family can prevent glitch propagation from an upset node. The logic family can be implemented in a standard, commercial CMOS process without any additional processing steps. The DC, transient, static power, upset recovery and layout characteristics of the new family, based on a commercial  $1\mu m$  CMOS N-Well process, were presented.

This logic family makes the design of completely SEU immune integrated circuits possible. The simulation results presented in this paper should prove useful to designers who need to implement SEU immune systems.

A test chip, which will be used to verify the simulations presented here, is currently being defined.

## Acknowledgement

The work reported here was supported in part by NASA under Space Engineering Research Center grant NAGW-1406. The author would also like to thank Sterling Whitaker, Don Thelen and Kelly Cameron, of the NASA SERC for VLSI System Design, for helping me understand the principles of ratioed logic design.

## References

- [1] D. Binder, E. Smith and A. Holman, "Satellite Anomalies from Galactic Cosmic Ray", *IEEE Transactions on Nuclear Science*, Vol. NS-22, No. 6, Dec. 1975, pp. 2675-2680.
- [2] J. Pickel and J. Blanford, "Cosmic Ray Induced Errors in MOS Memory Cells", *IEEE Transactions on Nuclear Science*, Vol. NS-25, No. 6, Dec. 1978, pp. 1166-1171.
- [3] W. Kolasinski, J. Blake, J. Anthony, W. Price and E. Smith, "Simulation of Cosmic Ray Induced Soft Errors and Latchup in Integrated Circuit Computer Memories",

- IEEE Transactions on Nuclear Science*, Vol. NS-26, No. 6, Dec. 1979, pp. 5087-5091.
- [4] J. Pickel and J. Blanford, "CMOS RAM Cosmic Ray Induced Error Rate Analysis", *IEEE Transactions on Nuclear Science*, Vol. NS-28, No. 6, Dec. 1981, pp. 3962-3967.
  - [5] E. Petersen, P. Shapiro, J. Adams and E. Burk, "Calculation of Cosmic Ray Induced Soft Upsets and Scaling in VLSI Devices", *IEEE Transactions on Nuclear Science*, Vol. NS-29, No. 6, Dec. 1982, pp. 2055-2063.
  - [6] J. Pickel, "Effect of CMOS Miniaturization on Cosmic Ray Induced Error Rate", *IEEE Transactions on Nuclear Science*, Vol. NS-29, No. 6, Dec. 1982, pp. 2049-2054.
  - [7] S. Diehl, A. Ochoa, P. Dressendorfer, R. Koga and W. Kolasinski, "Error Analysis and Prevention of Cosmic Ion Induced Soft Errors in Static CMOS RAMs", *IEEE Transactions on Nuclear Science*, Vol. NS-29, No. 6, Dec. 1982, pp. 2032-2039.
  - [8] J. Abraham, E. Davidson and J. Patel, "Memory System Design for Tolerating Single Event Upsets", *IEEE Transactions on Nuclear Science*, Vol. NS-30, No. 6, Dec. 1983, pp. 4339-4344.
  - [9] S. Cunningham, "Cosmic rays, Single Event Upsets and Things that Go Bump in the Night", *Advances in Astronautical Sciences*, Vol. 57, 1984.
  - [10] J. Browning, R. Koga and W. Kolasinski, "Single Event Upset Rate Estimates for a 16K CMOS SRAM", *IEEE Transactions on Nuclear Science*, Vol. NS-32, No. 6, Dec. 1985, pp. 4133-4139.
  - [11] S. Cunningham, "Living with Things that Go Bump in the Night", *Advances in Astronautical Sciences*, Vol. 56, 1985.
  - [12] R. Johnson and S. Diehl, "An Improved Single Event Upset Resistive Hardening Technique for CMOS Static RAMs", *IEEE Transactions on Nuclear Science*, Vol. NS-33, No. 6, Dec. 1986, pp. 1727-1733.
  - [13] A. Ochoa, C. Axness H. Weaver and J. Fu, "A Proposed New Structure for SEU Immunity in SRAM Employing Drain Resistance", *IEEE Electron Device Letters* Vol. EDL-8, No. 11, Nov. 1987, pp. 537-539.
  - [14] H. Weaver, C. Axness J. McBrayer, J. Fu, A. Ochoa and R. Koga, "An SEU Tolerant Memory Cell Derived from Fundamental Studies of SEU Mechanism in SRAM", *IEEE Transactions on Nuclear Science*, Vol. NS-34, No. 6, Dec. 1987, pp. 1281-1286.
  - [15] J. Rollins and J. Choma, "Single Event Upset in SOS Integrated Circuits", *IEEE Transactions on Nuclear Science*, Vol. NS-34, No. 6, Dec. 1987, pp. 1713-1717.
  - [16] L. Rockett, "An SEU Hardened CMOS Data Latch Design", *IEEE Transactions on Nuclear Science*, Vol. 35, No. 6, pp. 1682-1687, Dec., 1988.

- [17] S. E. Diehl and J. E. Vinson, "Considerations For Single Event Immune VLSI Logic", *IEEE Transactions on Nuclear Science*, Vol. NS-30, pp. 4501-4507, Dec., 1983.
- [18] J. F. Leavy, L. F. Hoffmann, R. W. Shovan and M. T. Johnson, "Upset Due to a Single Particle Caused Propagated Transient in a Bulk CMOS Microprocessor", *Proceedings of the Nuclear and Space Radiation Effects Conference*, July, 1991.
- [19] Y. Savaria, J. Hayes, N. Rumin and V. Agarwal, "A Theory for the Design of Soft-Error-Tolerant VLSI Circuits", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 1, pp. 15-23, Jan., 1986.
- [20] Y. Savaria, N. Rumin, J. Hayes and V. Agarwal, "Soft Error Filtering: A Solution to the Reliability Problem for Future VLSI Digital Circuits", *Proceedings of the IEEE*, Vol. 24, No. 5, pp. 669-683, May, 1986.
- [21] S. Kang and D. Chu, "CMOS Circuit Design for Prevention of Single Event Upset", Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1986.
- [22] S. Whitaker, J. Canaris and K. Liu, "SEU Hardened Memory Cells for a CCSDS Reed Solomon Encoder", *Proceedings of the Nuclear and Space Radiation Effects Conference*, July, 1991.
- [23] W. Nagel, "SPICE2: a Computer Program to Simulate Semiconductor Circuits", *ERL-M520*, Electronics Research Lab, University of California, Berkeley, May 1975.

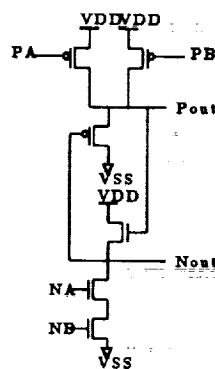


Figure 2: SEU Immune two-input NAND.

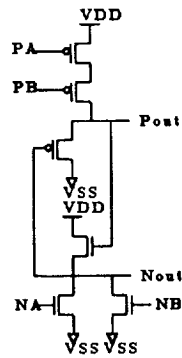


Figure 3: SEU Immune two-input NOR.

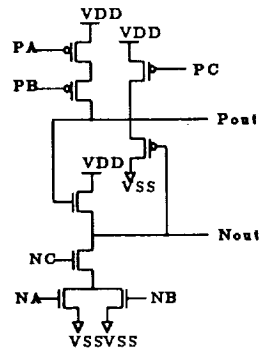


Figure 4: SEU Immune three-input OrNand.

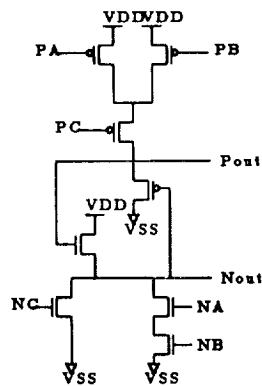


Figure 5: SEU Immune three-input AndNor



## Cellular Logic Array for Computation of Squares<sup>1</sup>

M. Shamanna, S. Whitaker and J. Canaris  
NASA Space Engineering Research Center  
for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract-** A cellular logic array is described for squaring binary numbers. This array offers a significant increase in speed, with a relatively small hardware overhead. This improvement is a result of novel implementation of the formula  $(x + y)^2 = x^2 + y^2 + 2xy$ . These results can also be incorporated in the existing arrays achieving considerable hardware reduction.

### 1 Introduction

The advent of VLSI has spurred a renewed interest in the development of specialized arithmetic circuits. Special arithmetic functions like squares and square-roots are generally implemented in software. However, when a machine is designed for a specific application, wherein squaring is a frequent process, it may prove advantageous in terms of speed to use a hardware implementation. Most of the approaches, reported in literature for squaring and square-rooting, use array multipliers or special purpose arrays which perform a multitude of other operations in addition to squaring. As a result, there are very few arrays which are solely devoted to extraction of squares. However, Dean[1] has reported such a dedicated array which is probably among one of the fastest squaring circuits known, thus far. In addition, Dean's array uses considerably less hardware than other arrays reported so far. Hence Dean's array has been selected as the obvious choice for comparison with the array proposed in this paper. The proposed array, will provide a significant gain in speed, with a very small hardware overhead, as compared to Dean's squarer[1].

### 2 Algorithm

Dean[1] has not presented a formal algorithm for his implementation. So, the widely used general binary squaring algorithm[3] will be presented first followed by the proposed algorithm for purposes of clarity and easy understanding. The existing algorithm for binary squaring is generally formulated as follows:

$$(1)^2 = (01)_b$$

$$(a_1 1)^2 = (a_1)^2 + (0a_1 01)_b \quad \text{or}$$

---

<sup>1</sup>This research was supported ( or partially supported ) by NASA under Space Engineering Research Center Grant NAGW-1406.

$$F_2 = F_1 + (0a_101)_b$$

where  $F_1 = (01)_b$  if  $a_1 = 1$  and  $F_1 = (00)_b$  otherwise. Similarly, we have

$$(a_2a_11)^2 = (a_2a_1)^2 + (00a_2a_101)_b, \text{ or}$$

$$F_3 = F_2 + (00a_2a_101)_b$$

In general if  $a_{r+1} = 1$  then,

$$F_{r+1} = F_r + D_r$$

where  $F_r = (a_r a_{r-1} \dots a_2 a_1)^2$  is the  $r^{\text{th}}$  square and  $D_r = \overbrace{00 \dots 0}^{r \text{ times}} a_r a_{r-1} \dots a_1 01$  is called the  $r^{\text{th}}$  radicand. It is obvious that  $F_{r+1} = F_r$  if  $a_{r+1} = 0$ . The above iterative formula applies for all  $r = 1, 2, \dots, n$ . Figures 4 and 5 show the schematic details of a three bit squaring array for the above algorithm[3].

The proposed algorithm makes use of the well known formula  $(x + y)^2 = x^2 + y^2 + 2xy$ . Consider a three bit number  $(a_2 2^2 + a_1 2^1 + a_0 2^0)$ . The LSB-1 and LSB of the square of any number will respectively be 0 and LSB of the original number itself. Therefore,

$$(a_2 2^2 + a_1 2^1 + a_0 2^0)^2 = (a_2 + a_1)2^4 + (a_2 a_0)2^3 + (a_1 a_0 + a_0)2^2 + a_0.$$

The same result can also be achieved by the repeated application of the formula  $(x + y)^2 = x^2 + y^2 + 2xy$  where  $y$  is the LSB and  $x$  is the rest of the binary number.

$$\begin{aligned} \underbrace{(a_2 2^1 + a_1 2^0)}_x^2 &= \underbrace{(a_2 2^1)^2}_{x^2} + \underbrace{2(a_2 a_1 2^1)}_{2xy} + \underbrace{a_1^2 2^0}_{y^2} \\ &= (a_2)^2 2^2 + (a_2 a_1) 2^2 + a_1^2 2^0 \\ &= (a_2 + a_2 a_1) 2^2 + a_1^2 2^0 \end{aligned} \quad (1)$$

Also,

$$\begin{aligned} \underbrace{(a_2 2^2 + a_1 2^1)}_x^2 + \underbrace{a_0 2^0}_y^2 &= \underbrace{(a_2 2^2 + a_1 2^1)^2}_{x^2} + \underbrace{2(a_2 a_0 2^2 + a_1 a_0 2^1)}_{2xy} + \underbrace{a_0^2 2^0}_{y^2} \\ &= (a_2 2^1 + a_1 2^0)^2 2^2 + (a_2 a_0 2^3 + a_1 a_0 2^2) + a_0^2 2^0 \end{aligned} \quad (2)$$

Equation 1 proves that the LSB-1 bit and the LSB of the final answer is always 0 and the LSB of the original number itself respectively. Since multiplication by 2 implies a left-shift by one bit position the term  $(2a_2 a_1)$  has been shifted from the  $2^1$  bit position to  $2^2$  bit position in Equation 1. This result for a three bit binary number is realized by the array of Figure 1. The algorithm can easily be extended to any  $n$  bit number. The novelty of the algorithm lies in the fact that squaring of the number is carried out in steps coupled with the ingenious use of left-shifts in the bit positions.



### 3 Comparison

The implementation of the proposed algorithm for a 3 bit and a 4 bit number has been illustrated in Figures 1 and 3 respectively. The proposed array is built of the basic half-adder cell shown in Figure 2. Its function may be defined as follows:

$$u = (w + v_{-1}) \oplus (xy)$$

$$v = (w + v_{-1}) \cdot (xy)$$

The symbols  $+$  and  $\cdot$  stands for the Inclusive-Or and And operations in the above expressions.

The implementation of 3 bit squarer based on Dean's algorithm is also illustrated in the Figures 6 and 7. The basic cell (Figure 7) has two control inputs  $A$  and  $B$ . The inputs on the lines  $C$  and  $D$  are added in the cell,  $S$  being the sum out and  $P$  being the carry out. When both  $A$  and  $B$  are present, a further digit is added to the sum (and carry), so that the cell then functions as a full-adder[1].

It can be seen that the proposed array has  $1 + \sum_{i=3}^n i$  whereas Dean's array [1] uses  $1 + \sum_{i=1}^n i$  cells resulting in a overhead of  $(n - 2)$  cells. However, the hardware inside the proposed basic cell is much simpler, as it utilizes only half-adders, compared to full-adders in Dean's array. So the increase in the number of cells is offset by the reduction in the complexity of the individual cell. This leads to the authors contention that the hardware overhead which translates into increased chip area is almost negligible. Moreover, the propagation time through the proposed array is only  $n\tau$  as compared to  $(2n - 3)\tau$  which is the delay through Dean's array. The hardware overhead-speed gain relation follows the square law for most specialized arithmetic arrays. Here, an increase in speed has been accomplished with a linear increase in hardware.

The proposed array has a number of unused inputs which can be used to add in an other number so that the array would function as a full squarer (all outputs in 1 state). A specialized array of this sort has a number of applications including the generation of binary logarithms[2] which depends on iterative squaring.

### 4 Conclusions

A new cellular array for extraction of squares of binary numbers has been presented. An squaring algorithm based on the formula  $(x + y)^2$  has been described. The proposed array provides impressive speed gains compared to the existing arrays at the expense of negligible hardware overhead. It is hoped, that the algorithm discussed in this paper will provide fresh insights, to reduce redundant hardware present in most of the existing squaring arrays.

### References

- [1] K. J. Dean, Cellular Logical Array for Obtaining the Square of a Binary Number, *Electronics Letters*, Vol. 5, Aug. 1969, pp.370-371.

#### 2.4.4

- [2] K. J. Dean, A fresh approach to logarithmic computation, *Electron. Engng.*, 41, April 1969, pp.488-490.
- [3] K. Hwang, Computer Arithmetic: Principles, Architecture and Design, *John Wiley and Sons*, 1979.

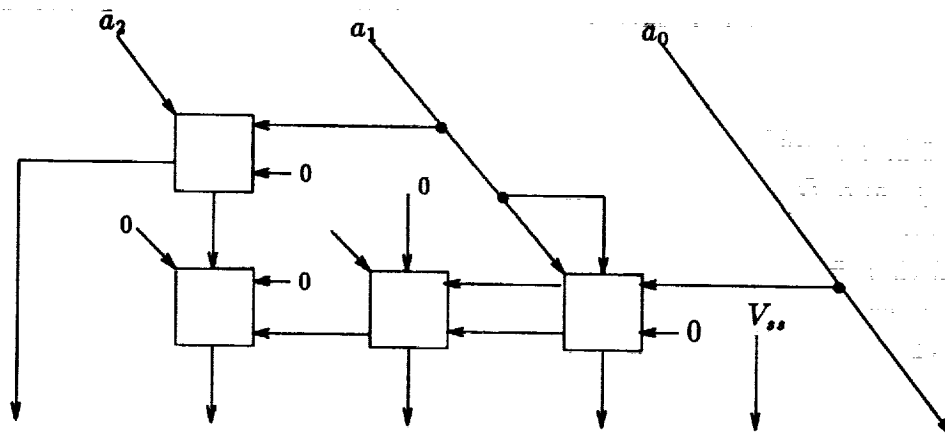


Figure 1: Proposed squaring array for three bit numbers

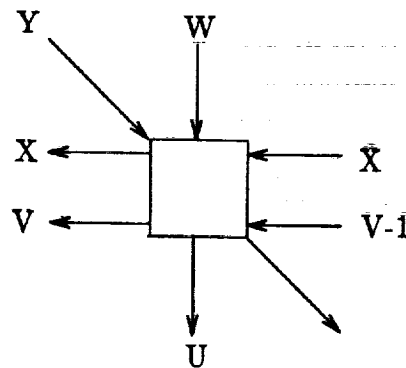


Figure 2: Basic cell used in the proposed squaring array

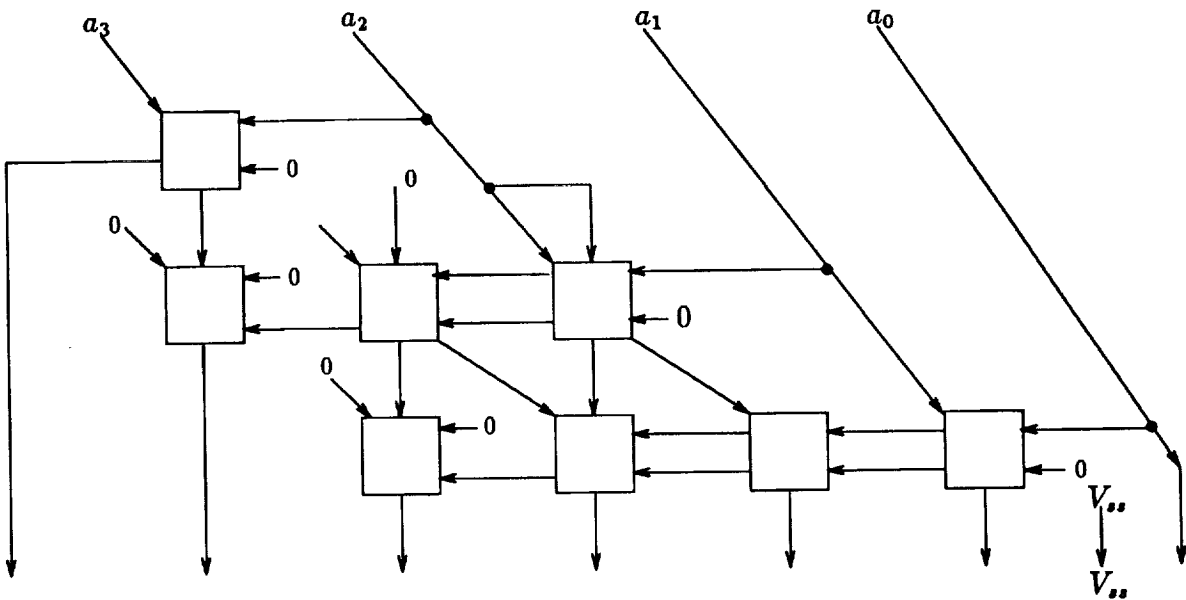


Figure 3: Proposed squaring array for four bit numbers

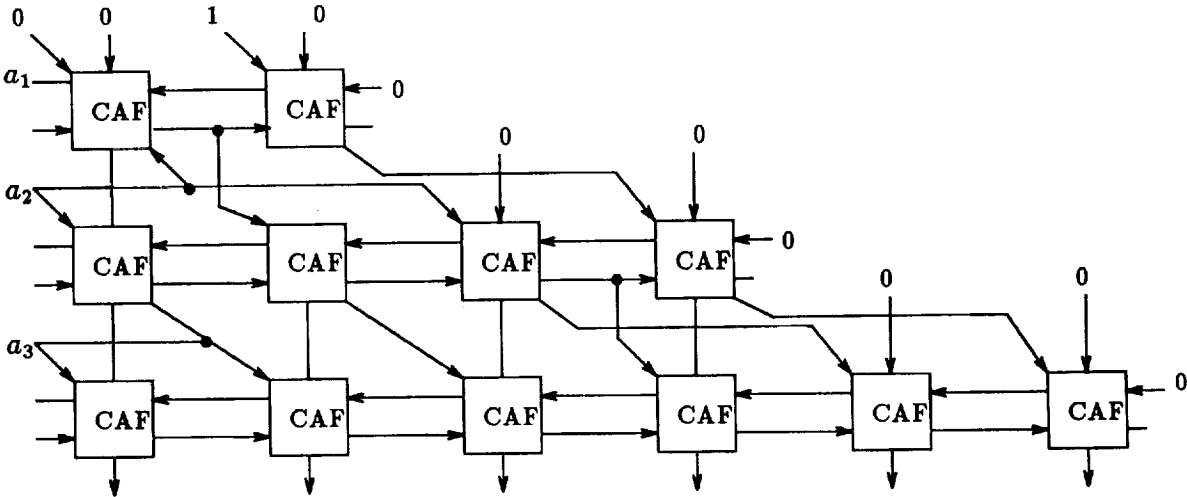


Figure 4: A three bit squaring array using the general algorithm

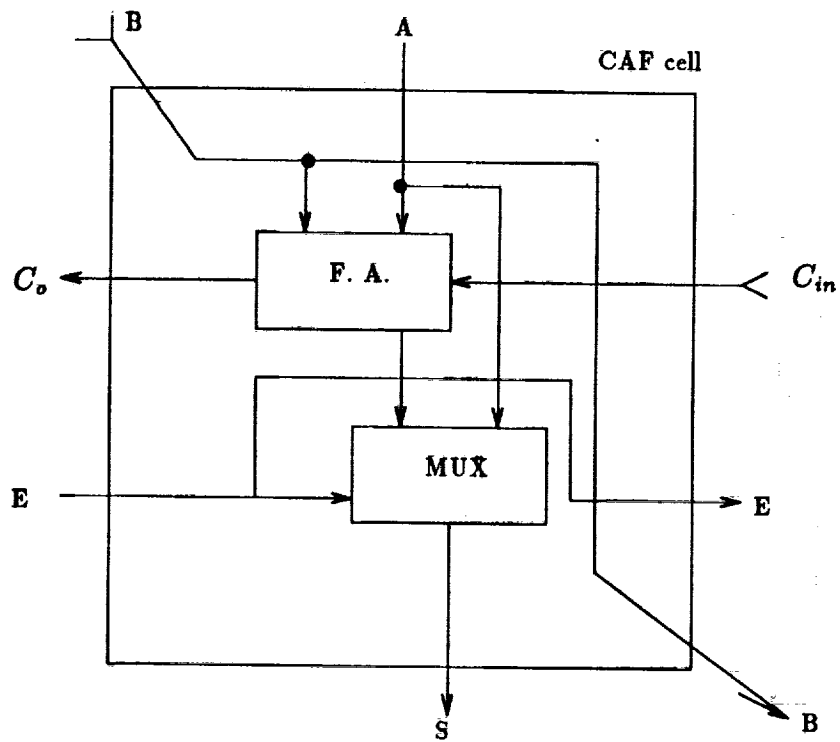


Figure 5: Basic cell used in the general three bit squaring array

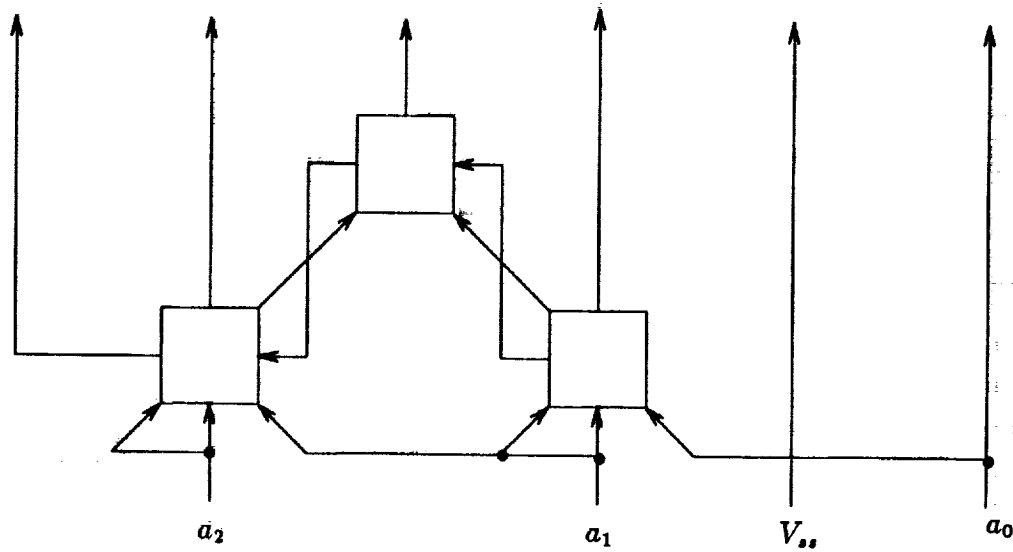


Figure 6: Dean's array for three bit numbers

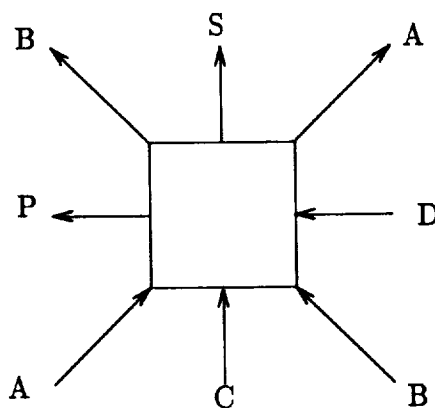


Figure 7: Basic cell used in Dean's array



# Fault Tolerant Sequential Circuits Using Sequence Invariant State Machines

M. Alahmad and S. Whitaker  
NASA Space Engineering Research Center  
for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract** - The idea of introducing redundancy to improve the reliability of digital systems originates from papers published in the 1950s. Since then, redundancy has been recognized as a realistic means for constructing reliable systems. This paper will introduce a method using redundancy to reconfigure the Sequence Invariant State Machine (SISM) to achieve fault tolerance. This new architecture is most useful in space applications, where recovery rather than replacement of faulty modules is the only means of maintenance.

## 1 Introduction

Fault tolerance is essential feature for digital systems where reliability, availability and safety are of vital importance. Such systems include aerospace missions, where a recovery procedure must be employed as means of maintenance, rather than replacement procedures which would be impossible during such missions.

Most digital systems can be divided into two functional blocks: the controller and the data path. The controller is a sequential circuit that performs certain tasks based on external and internal information. A programmable hardware architecture has been developed that enables a controller's hardware to be designed without a knowledge of the exact sequence of the input data to be incorporated [1]. This programmable architecture is called a Sequence Invariant State Machine (SISM).

This paper will introduce a method to achieve fault tolerance in the SISM design using dynamic redundancy. With this method, faulty controllers can recover and resume operation. Two different architectures are proposed and analyzed in terms of transistor count, size and fault detection. One architecture is clearly superior to the other.

## 2 SISM Overview

With the SISM realization, any flow table can be implemented without a change in the hardware configuration. That is given the number of states  $m$  and the number of inputs  $n$ , a hardware circuit is easily derived, that can implement any sequence of states.

	$I_1$	$I_2$	$I_3$
A	C, 1	B, 1	A, 0
B	D, 0	C, 1	B, 0
C	E, 0	D, 0	C, 0
D	F, 1	E, 1	D, 1
E	A, 0	F, 0	E, 1
F	B, 0	A, 1	F, 1

Table 1: General 6-states, 3-input flow table.

$y_1$	$y_2$	$y_3$		$I_1$	$I_2$	$I_3$
0	0	0	A	0 1 0, 1	0 0 1, 1	0 0 0, 0
0	0	1	B	0 1 1, 0	0 1 0, 1	0 0 1, 0
0	1	0	C	1 0 0, 0	0 1 1, 0	0 1 0, 0
0	1	1	D	1 0 1, 1	1 0 0, 1	0 1 1, 1
1	0	0	E	0 0 0, 0	1 0 1, 0	1 0 0, 1
1	0	1	F	0 0 1, 0	0 0 0, 1	1 0 1, 1
1	1	0	G	0 0 0, 0	0 0 0, 0	0 0 0, 0
1	1	1	H	0 0 0, 0	0 0 0, 0	0 0 0, 0

Table 2: State Assignment and Redundant States for Table 1.

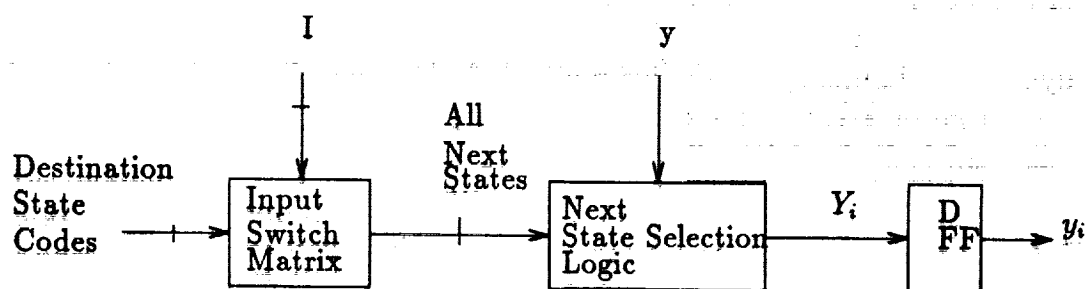


Figure 1: General SISM Architecture.



Table 1 shows a general 6 states, 3-input flow table. The state assignment for this table is shown in Table 2. Figure 1 shows the SISM architecture for one of the next state variables in Table 2. There are two identical architectures for the remaining two variables. Only the destination state codes are different. The Figure consist of the following components.

- The destination state codes which are derived from the next state entries in the state assignment table by inspection. For example, the destination state codes for state B and state variable  $y_i$  are the next state bits  $Y_i$  associated with state B. Therefore, the destination state codes for state B are (000, 110, 101) under input states ( $I_1; I_2; I_3$ ) and variables ( $y_1; y_2; y_3$ ) respectively.
- The input switch matrix which is combinational logic that produces all the possible next state entries for each current input state.
- The next state logic which consists of an independent path for each of the present states in the state assignment flow table.
- The storage element, a D-FF, that preserves the present state.

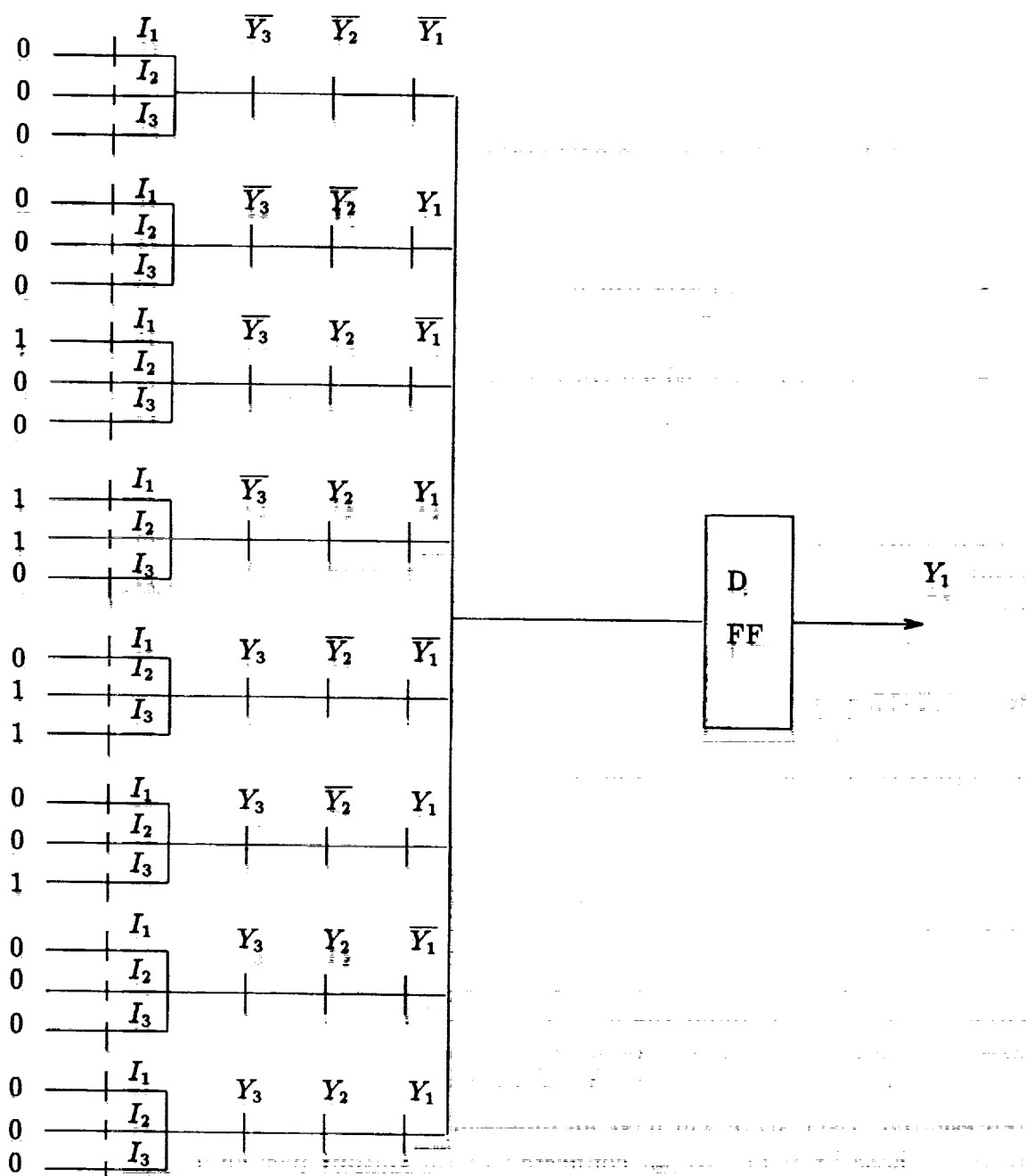
The current input state selects the set of potential next states that the circuit can assume (input column in the flow table). The present state variables select the exact next state (row in the flow table) that the circuit will assume at the next clock pulse.

### 3 SISM Implementation

Two pass transistor networks which make the SISM fault tolerant will next be discussed and compared in terms of space and the number of transistors. The input switch matrix is shown in both structures as a logic block, since it is identical in both designs.

#### 3.1 FCS Design

A Fully Coded Structure (FCS) [4] network is defined as a fully specified pass network circuit. A knowledge about the number of next state variables is sufficient to achieve this design. Thus, the FCS is a design by inspection. Using Table 2 as a reference, three state variables are required to implement this table. Therefore, eight unique states can be represented. Each state will have an independent branch with all the variables as control terms. Those branches are all connected to the output pass function. Only one branch is activated by any combination of control variables at a given time, since each branch is encoded uniquely. The output pass function is the logical OR or the summation of all states. Figure 2 shows the complete FCS structure for the next state variable  $Y_1$  in Table 2. The other two variables have identical structure, but different destination state codes.



**Figure 2: Structure of the next state variable  $Y_1$  using the FCS structure.**

### 3.2 BTS Design

A Binary Tree Structure (BTS) [3] network is defined as a pass network in which exactly two branches join at every node and the control term of one branch is the complement of the control term of the other branch. Generally, each control term is a single control variable and the number of nodes exceeds one. A BTS network is constructed by partitioning each next state variable in a specific manner until all the variables have been partitioned. Therefore, a BTS network is also designed by inspection.

Consider the flow table shown in Table 1. Three variables are needed to implement this flow table. The procedure is general and can be applied to any state machine. Firstly, start with the output node and partition the variable  $Y_3$  into two branches. One of the branches will have  $Y_3$  as the control variable and the other branch will have  $\overline{Y_3}$  as the control variable. Secondly, for each node at the end of each of the newly constructed branches, construct two more branches for the control variable  $Y_2$  and its complement. Thirdly, for each node at the end of the new branch, construct two branches for the variable  $Y_1$  and its complement. With this step the design structure is completed. Figure 3 shows the complete BTS structure for the next state variable  $Y_1$ . The other two next state variables are identical in structure and only the destination state codes are different.

### 3.3 Comparison

The BTS and FCS structures both use pass transistor networks. The number of transistors in the BTS structure is less than the number of transistors in the FCS structure, since the BTS structure is partitioned around each control variable. In terms of space and size, the BTS would appear to require less space. However, using the SISM compiler developed by Buehler [2] to design the BTS structure, the space required for each design is basically the same. The extra space available in the BTS structure is difficult to utilize. Using the SISM compiler, a custom drawn SISM layout for one of the variables in Table 2, using both structures, is shown in Figures 4 and 5.

### 3.4 Destination State Codes Implementation

The destination state codes are all the inputs that must be fed to either the BTS or the FCS structure in order to implement a state table. The inputs can be driven in several ways. They could be directly connected to VDD/VSS or they could be driven by the output of a shift register. The input array could also be constructed as a programmable memory such as EPROM.

In order to achieve programmability in the SISM structure, the data must not be hard-wired. If data were implemented using VDD and VSS connections then, the programmable nature of this design is limited to single mask programmability. Using a shift register will achieve the programmability objectives. The shift register will, however, increase the size of the circuit. If the EPROM is implemented on the IC, the size of the controller will also increased but since an EPROM cell is considerably smaller than a D-FF, the size impact

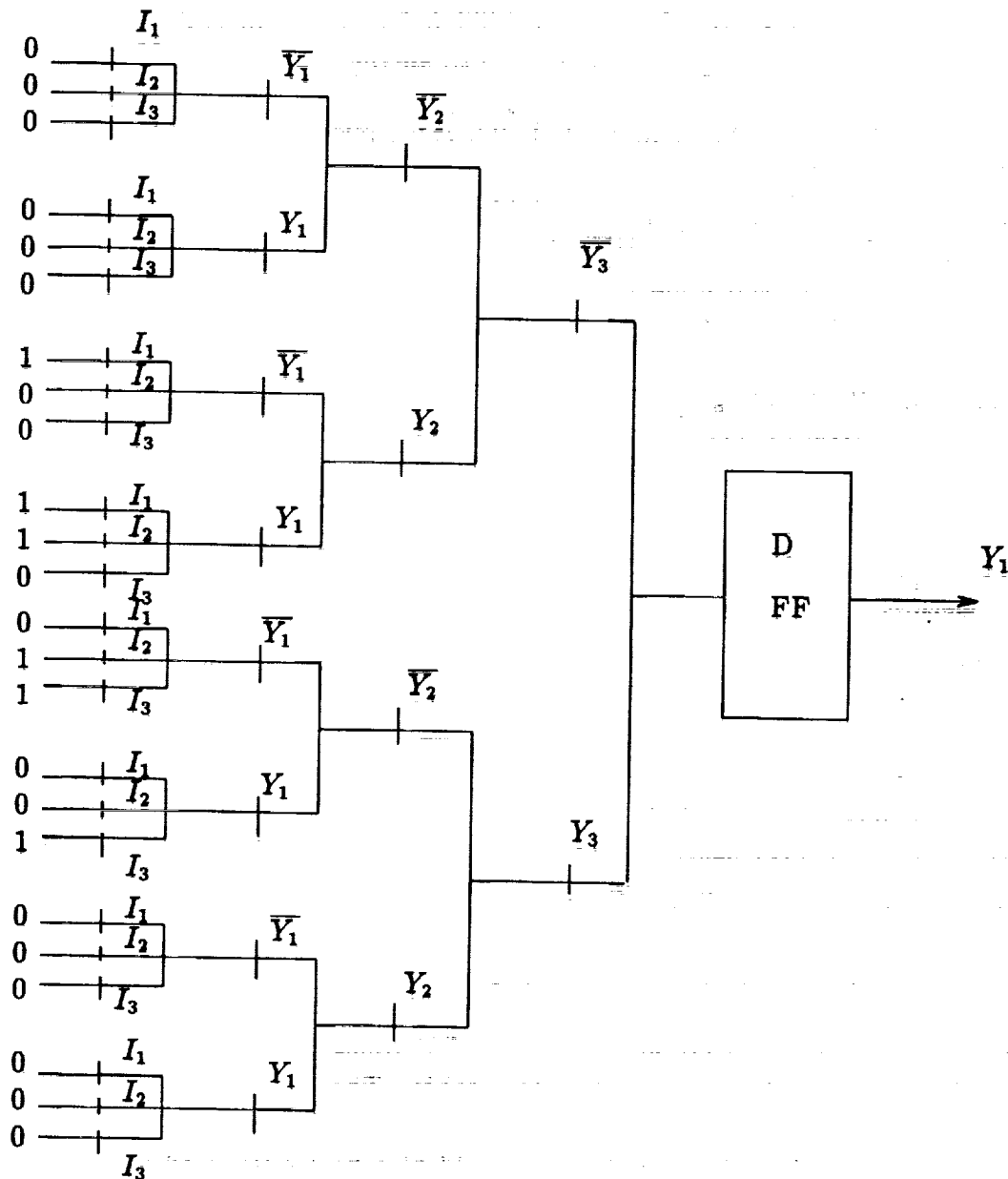


Figure 3: Complete structure for the variable  $Y_1$  using the BTS architecture.

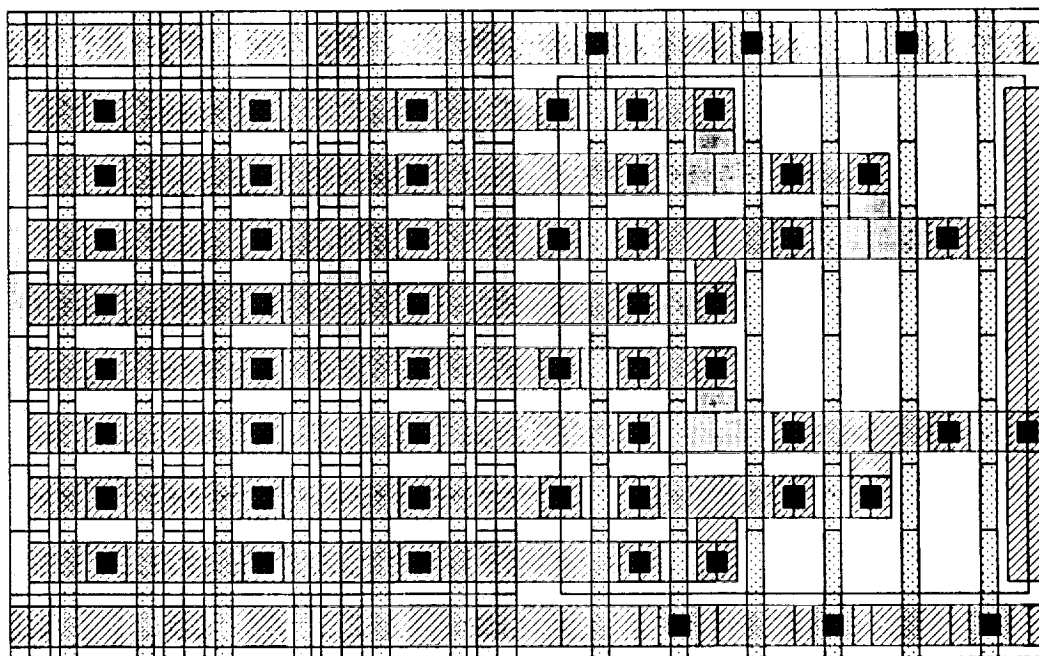


Figure 4: SISM layout using the BTS structure.

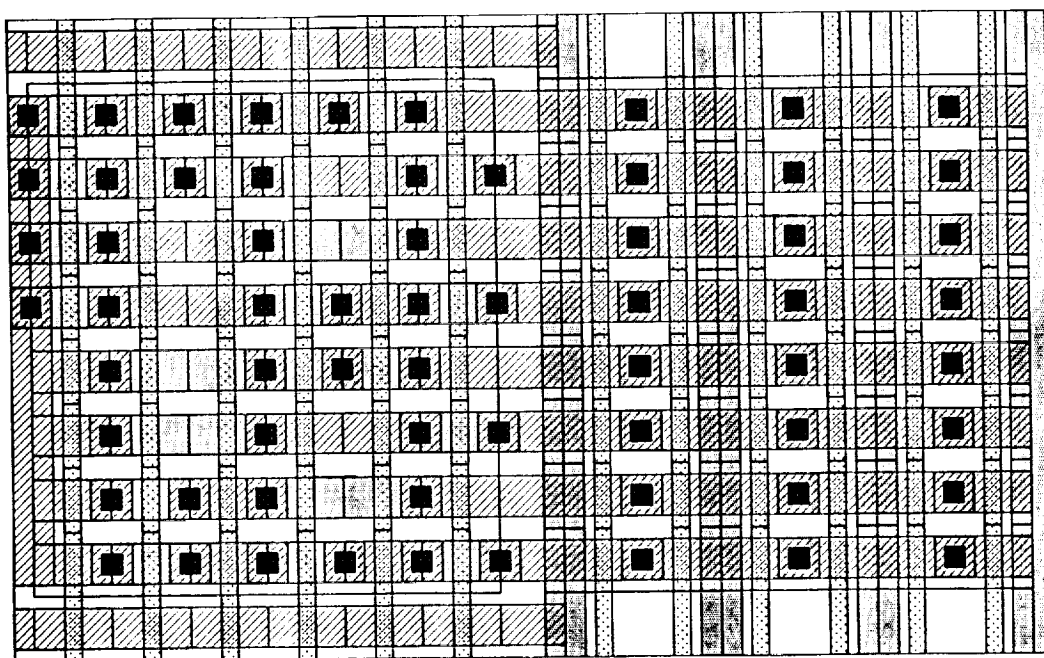


Figure 5: SISM layout using the FCS structure.

	$I_1$	$I_2$	$I_3$
A	C	B	A
B	D	C	B
C	E	D	C
D	F	E	D
E	A	F	E
F	B	A	F
G	A	A	A
H	A	A	A

Table 3: Fully specified flow table.

is much less than that for the shift register approach:

## 4 Achieving Fault Tolerance

In incorporating fault tolerance in any digital system, two approaches can be considered. The first approach is called static redundancy, also known as fault masking, which uses extra components such that the effect of a faulty component is masked instantaneously. The second approach is called dynamic redundancy, which has extra components but only one component operates at a time. If a fault is detected in the operating module, it is switched out and replaced by a spare. This dynamic redundancy requires consecutive actions of fault detection and fault recovery [5].

The idea of dynamic redundancy to achieve fault tolerance can be applied to the SISM structure. Hence, the operating module refers to all the paths (states) in the next state selection logic that construct the state machine. And the spare parts refer to the unutilized logic (redundant states) in the architecture. Therefore, if a fault has been detected in a given state (i.e. the path that identifies that state), a spare path is switched to replace the current path and correct operation is resumed.

Most state machines do not utilize all available states. Therefore, some of those states can be thought of as spare states and are redundant. To optimize the versatility and robustness of a controller, the redundant states can be used to replace any state which exhibits a malfunction. By applying a method for reconfigurability, the redundant states can be used to improve the reliability and to enhance the performance of an IC.

With reference to Table 1, there are six states, therefore three variables are needed to implement this flow table. With three variables, a maximum of eight states are available. Six of these states are used and two states are redundant. However, the next state entries for each of the two redundant states have been assigned the initial value (which is a safe output in all cases) as shown in Table 3, with the assumption that state A is the initial state. If state B tested faulty, then one of the redundant states, such as state G, could be used to replace state B to achieve correct operation.

Both the BTS and the FCS will have extra logic, and the reconfigurability method can

be applied to use the extra logic. However, the location of a fault in the BTS can limit the use of the redundant logic and therefore decrease fault tolerance. That is, if a fault affects any of the transistors controlling  $Y_1$  or its complement in Figure 3, then the method is valid and redundant logic can be used to replace that faulty branch. However, if a fault affects  $Y_3$  or its complement then, there is not enough redundant logic to replace the entire faulty section. Therefore, the redundant logic has limited capabilities in the BTS structure. An identical structure can be added, but in doing so static redundancy can be achieved easily, at the cost of increasing the structure size by a factor of two.

The FCS structure, possesses a good structure. If any s-a-fault or s-op faults occur at the input or in the structure, then only one path (state) is effected. However, if a stuck-on faults occur in the structure, then two paths (states) will be affected at most. For example if a stuck at fault affects state B, then only the path that represents state B is affected and can be replaced. However, if a stuck-on affects state B, then two paths will be enabled at the same time. Therefore, the redundant logic can be used to replace this malfunction state. Hence the FCS structure is more applicable if dynamic redundancy is to be used.

Furthermore, the redundant logic in the FCS structure does not mask any of the faults that could occur in the structure. The reason being that the redundant logic does not replicate any of the existing states. Therefore, a fault in the structure or even in the redundant logic itself is testable.

## 5 Design Procedure

If any path in the FCS architecture becomes faulty due to the input being stuck at 1 or stuck at 0, a stuck open or shorted pass transistor, or any other malfunction, then the entire path is no longer correct and therefore must be replaced or recovered. To achieve fault tolerance, three methods must be used. They are error detection, fault location, followed by replacement and recovery. The primary concern is with the replacement and recovery technique. Once the designer has concluded that an error has occurred in a part of the IC, fault detection and location techniques are then applied to detect and locate the faulty part. If the faulty part is in the controller section of the circuit, then it must be determined where the fault has occurred, and the kind of fault that occurred.

Referring to Table 3, assume that the fault diagnosis has shown that state B is a faulty state. This corresponds to the path  $(\bar{Y}_3; \bar{Y}_2; Y_1)$  in Figure 2, then the following steps are applied.

### **STEP1**

Examine the flow table at hand and determine which of the redundant states will be used to replace state B. Since this flow table has two redundant states, State G is chosen. State H could have just as validly been chosen, but for simplicity the next state in order was chosen. Hence state G,  $(Y_3; Y_2; \bar{Y}_1)$  is chosen to replace state B.

### **STEP2**

Modify the flow table to reflect the new changes. That is scan the flow table and replace each next state entry of B with the new state G. Therefore, every where in the next state

	$I_1$	$I_2$	$I_3$
A	C	G	A
B	D	C	G
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	A	A	A
H	A	A	A

Table 4: Second step in the replacement procedure.

	$I_1$	$I_2$	$I_3$
A	C	G	A
B	D	C	G
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	D	C	G
H	A	A	A

Table 5: Third step in the replacement procedure.

entry of the state table, replace B with a G. Table 4 reflects this replacement process.

**STEP3**

Fill the next state entry of state G with the same next state entry as that of state B. That is the next state entries for G will be the same next state entries for B providing that step2 was completed. Table 5 shows the result of this step.

**STEP4**

The next state entries for state B are modified in such a way that masks the kind of permanent fault in the hardware.

1. If a stuck at fault, s-op or s-on faults occur at the input of the destination state codes or in the input switch matrix or a s-a-1 or s-a-0 fault on the destination codes, then disabling the B state is sufficient.
2. If a s-op is occurred in any of the variables, then the path is already disabled.
3. If a s-on fault occurs in any of the variables, then the destination state codes to the faulty path must be identical to those of the new path the fault assumes. That is, if the variable  $\overline{Y}_3$  in state B is stuck on, then this state becomes  $(1; \overline{Y}_2; Y_1)$  which is the same as state F. Therefore, the next state entries of state B must be the same as that of state F. Hence, when state F is enabled, state B is also enabled. To achieve



	$I_1$	$I_2$	$I_3$
A	C	G	A
B	G	A	F
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	D	C	G
H	A	A	A

Table 6: modified flow table.

$y_1$	$y_2$	$y_3$		$I_1$			$I_2$			$I_3$		
0	0	0	A	0	1	0	1	1	0	0	0	0
0	0	1	B	1	1	0	0	0	0	1	0	1
0	1	0	C	1	0	0	0	1	1	0	1	0
0	1	1	D	1	0	1	1	0	0	0	1	1
1	0	0	E	0	0	0	1	0	1	1	0	0
1	0	1	F	1	1	0	0	0	0	1	0	1
1	1	0	G	0	1	1	0	1	0	1	1	0
1	1	1	H	0	0	0	0	0	0	0	0	0

Table 7: Modified flow table.

correct operation, both states must have the same next state entries. As a result, the fault is masked. Table 6 shows the resulting flow table.

### **STEP5**

The new state assignment is then reflected in the modified flow table. Table 7 shows the state assignment and the next state entries assignment.

### **STEP6**

The destination state codes derived from the modified flow table determine the new data entries for the shift register.

With the completion of Step6, the operation of the circuit can be resumed with the same expected results.

Two final points are worth discussing. Firstly, if the state machine does utilize all of its states then an additional state variable must be added to allow this procedure to be employed. In order to demonstrate the procedure, the flow table shown in Table 8 is considered. As can be seen there are no extra states. Therefore, a new state variable is added and then the state assignment is revisited during the initial design to achieve redundancy. The next state equations and the hardware implementation will reflect this modification. The modified flow table is shown in Table 9.

Secondly, this method can be extended to achieve fault tolerance in the remaining parts of the circuit. This would be achieved by determining the faulty part and reconfiguring

$Y_1$	$Y_2$		$I_1$	$I_2$
0	0	A	D	B
0	1	B	C	C
1	0	C	A	D
1	1	D	B	A

Table 8: General 4-states, 2-input flow table.

$Y_1$	$Y_2$	$Y_3$		$I_1$	$I_2$
0	0	0	A	D	B
0	0	1	B	C	C
0	1	0	C	A	D
0	1	1	D	B	A
1	0	0	E	A	A
1	0	1	F	A	A
1	1	0	G	A	A
1	1	1	H	A	A

Table 9: Modified flow table.

the state machine in such a way as not to enable the faulty part, and to activate another part to replace it.

## References

- [1] S. Whitaker, S. Manjunath and G. Maki, "Sequence Invariant State Machines", *IEEE Journal of Solid State Circuits*, Vol. SC-26, Aug. 1991, pp .
- [2] David M. Buehler, "Sequence Invariant State Machine Compiler", Master Thesis, Dept. of Elect Engr., University of Idaho, Moscow, Idaho, Dec. 1990.
- [3] G. Peterson and G. Maki, "Binary Tree Structured Logic Circuits: Design and Fault Detection", *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, NY, Oct., 1984, pp. 671-676.
- [4] D. Radhakrishnan and G. Maki, *Digital Systems Design*, EE 440 Lecture Notes, University of Idaho, Fall 1989.
- [5] Parag K. Lala, *Fault Tolerant & Fault Testable Hardware Design*, Prentice-Hall International, Inc., London 1985.

# VLSI Architectures for Geometrical Mapping Problems in High-Definition Image Processing

K. Kim

Superconducting Super Collider Lab.  
2550 Beckleymeade Avenue  
Dallas, TX 75237

J. Lee

Department of Electrical Engineering  
University of Houston  
Houston, TX 77204-4793

**Abstract-** This paper exploits a VLSI architecture for geometrical mapping address computation. The geometric transformation is reviewed under the field of plane projective geometry, which evokes a set of basic transformations to be implemented for the general image processing. The homogeneous and 2-Dimensional cartesian coordinates are employed to represent the transformations, each of which is implemented via an augmented CORDIC as a processing element. A specific scheme for a processor, utilizing fully-pipelining at the macro-level, parallel constant-factor-redundant arithmetic and fully-pipelining at the micro-level, is assessed to produce a single chip VLSI for the HDTV applications under the current state-of-art MOS technology.

## 1 Introduction

Geometrical transformations are widely discussed in the field of digital image processing such as high-definition television(HDTV), image recognition, interactive computer graphics and vision processing [1,2,3]. The primary interest of these transformations is to project an image in a different domain, to extract additional signal conveying the information of the image. Moreover, it affords value-added images over the conventional displaying via the high resolution, definition, and flexible framing. Consequently, a geometrical mapping processor is about to appear to support a real-time processing. In recent years, several geometrical mapping processing modules have been developed and applied successfully for an appropriate application. They are implemented either by popular graphics package or application software accompanying an acceleration box [5], or a VLSI Processor [6]. We are interested in a VLSI implementation of a processor to realize a real-time speed for TV image processing, with a sufficient set of transformations to make a value-added display.

It has been known that two barriers have existed toward the development of such a processor. The first is the lack of a sufficiently high-speed arithmetic computation technique to generate the mathematical functions required for geometrical mapping. The second is the need for an extensive library of geometrical mapping functions. To overcome these, two key techniques have been developed in [4,6]: The first is a very high speed radix-2

signed-digit adder and the second is a pipelined micro-programmable arithmetic function generator. In this paper, we study the same problem with the goal of optimizing the overall functionality and performance. We achieve this goal by improving the basic cell.

In the following section, we will review the requirement of the geometrical mapping processor by introducing its definition and applications. In Section 3, we will study various CORDIC schemes to implement a basic cell, which can be used to compose the necessary function set for the geometric transformations.

## 2 Geometrical Mapper

Transformation of a sub-image requires a mapping of the sub-image from one point to the transformed, pixel by pixel. To rearrange the image, it is necessary to calculate the destination address of each pixel, which is called a geometrical mapper.

In the field of plane projective geometry, transformation from a point to another point is represented as a multiplication in homogeneous coordinates [10]. Let a 2-dimensional (2-D) point  $p_x = (x, y)$  is represented as  $(ax, ay, a)$  in right-handed homogeneous coordinates, with a non-zero constant  $a$ . The vector  $p_x$  is referenced to an origin  $(0, 0)$ . The most useful transformations are translation, scaling and rotation, examples of which are respectively defined as:

$Trans(x, d)$  : translating  $p_x$  to  $(x + d, y)$

$Rot(x, \theta)$  : rotating the vector  $p_x$  by an angle of  $\theta$  about x-axis

$Scale(x, c)$  : scaling the vector  $p_x$  by  $c$  along x-axis.

$$\begin{aligned}(x, y) \cdot Trans(x, d) &= (x + d, y) \\ (x, y) \cdot Rot(x, \theta) &= (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta) \\ (x, y) \cdot Scale(x, c) &= (cx, y)\end{aligned}\tag{1}$$

Or, the composite of 3 different transformations in 2-D is represented by

$$T = \begin{vmatrix} c \cdot \cos \theta & \sin \theta & 0 \\ -c \cdot \sin \theta & \cos \theta & 0 \\ cd & 0 & 1 \end{vmatrix},\tag{2}$$

which is called an affine transformation. The affine transformation is performed via a set of multiplication and trigonometric function.

Easily observed, the affine transformation is a necessary transformation to map a sub-image into another area of the image domain, with sliding, re-sizing and proper rotation. Its immediate applications include sub-image generation for the multiple picture-in-picture (PIP) TV, image template generation for the recognition and vision/graphics processing.

Further sophisticate transformation useful for the general image processing is the spherical, which basically transforms between the plane and sphere surfaces. A spherical transformation from  $p_x$  to  $q_x = (u, v)$  can be represented by using a set of elementary functions,

such as square root, division, and squaring operations.

$$\begin{aligned} u &= \frac{rx}{\sqrt{r^2 - x^2 - y^2}} \\ v &= \frac{ry}{\sqrt{r^2 - x^2 - y^2}}, \end{aligned} \quad (3)$$

where  $r$  denotes the curvature degree of sphere surface. A conventional way to implement the transformations starts from a software package, i.e., interactive graphics package. To implement a dedicate hardware, possibly a set of modular structures in VLSI, it is necessary to figure out a basic cell of those functions, and there has been two different approach: the first based on a set of elementary function generators and the second on a programmable module. For the first approach, fast function generators are necessary and the performance is limited by the slowest function generator. Apparently, the trigonometric functions are the bottleneck while being implemented via the first idea. To optimize the trigonometric function generation, while considering the regularity of its structure, CORDIC has been suggested the recursiveness of the CORDIC iteration has been misleading a concept that the second approach is not usually better than the first one.

Recently, as VLSI technologies evolve, the effectiveness of the integration is not simply a complexity of the multiplication but also implies a communication complexity more than the multiplication complexity include regularity of the structure, simplicity of the design and localization of the interfacing. In these senses, CORDIC has been widely reviewed again, and shown to be appropriate for a couple of algorithmic processors. In brief, CORDIC is a set of recursive algorithms, which can be easily programmed to generate a set of elementary functions via a different mode and a proper zero-enforcing. It is also capable of vector-oriented processing.

### 3 CORDIC Techniques

In this section, we will review CORDIC functions to i) perform a vector transformation and ii) generate elementary functions. CORDIC comprises of three linear recursive equations, namely  $X$ -,  $Y$ - and  $Z$ - recurrences. Table 1 summarizes the computing mode, input and output specifications of CORDIC functions of our interest. As shown in the Table, these functions are classified into two cases, one which enforces  $Z[N]$  to be zero (known as *rotating*) and the other which enforces  $Y[N]$  to be zero (known as *vectoring*). We will discuss these cases in the following sections.

#### 3.1 Rotating case

The vector rotation for  $p_x = (X[0], Y[0])$  by the angle  $\theta$  can be realized by an iteration algorithm called CORDIC [12] instead of computing trigonometric functions and applying matrix multiplication. CORDIC realizes a vector rotation by a partial sum of micro-angle rotations with a pre-fixed sequence of angles. When the rotation macro-angle is represented

Mode	Input	Enforcing	Output
Circular	$Z[0] = \theta, (X[0], Y[0])$	$Z[N] = 0$	Rotation by $\theta$
Circular	$Z[0] = 0, (X[0], Y[0])$	$Y[N] = 0$	$X[N] = \sqrt{X[0]^2 + Y[0]^2}$ $Z[N] = \tan^{-1}(Y[0]/X[0])$
Linear	$Z[0] = 0, (X[0], Y[0])$	$Y[N] = 0$	$Z[N] = Y[0]/X[0]$
hyperbolic	$(X[0], Y[0])$	$Y[N] = 0$	$X[N] = \sqrt{X[0]^2 - Y[0]^2}$

Table 1: Available CORDIC Processing

as a sum of decomposed micro-angles, i.e  $\theta = \sum_{k=0}^n \theta_k$ ,

$$r_x^t = \prod_{k=0}^n k_k \begin{bmatrix} 1 & -\tan\theta_k \\ \tan\theta_k & 1 \end{bmatrix} p_x^t \quad (4)$$

where  $k_k = \cos\theta_{i,k}$  is a micro-scale composing a final scale factor, explained later. Such a specific form of the pre-fixed micro-angle sequence as  $\tan^{-1} 2^{-i}$ , is attractive for VLSI implementation since it is composed only of additions, shiftings, and a arctangent lookup table

**Non-redundant :** The micro-iterations of the conventional (hereafter, it will be called non-redundant ) CORDIC use the following 3 linear recursive equations [12]:

$$\begin{aligned} X[i+1] &= X[i] + m\sigma_i 2^{-i} Y[i] \\ Y[i+1] &= Y[i] - \sigma_i 2^{-i} X[i] \\ Z[i+1] &= Z[i] - \sigma_i \tan^{-1} 2^{-i} \end{aligned} \quad (5)$$

where  $m$  will be set to one for the circular CORDIC, while  $m = 0$  for the linear and  $-1$  for the hyperbolic. With an initial value of  $Z[0] = \theta$ , CORDIC rotates initial values of  $X[0]$  and  $Y[0]$ , to the last value  $X[n]$  and  $Y[n]$  while making  $Z[i]$  close to zero in each  $i$  iteration, so that  $Z[n]$  is forced to be zero. With  $n$  number of iterations,  $n$ -bit accuracy of  $X[N]$  and  $Y[N]$  can be achieved. For a known angle, the direction of the rotation,  $\sigma_i$  can be pre-computed or calculated one by one on-the-fly using the following selection function.

$$\sigma_i = \begin{cases} 1 & \text{if } Z[i] \geq 0 \\ -1 & \text{if } Z[i] < 0 \end{cases} \quad (6)$$

The CORDIC rotation does not preserve the input norm. To get a rotated vector having the same length as the input  $(X[0], Y[0])$ ,  $X[n](Y[n])$  needs to be compensated by a scaling factor  $K$

$$K = \frac{\| [X[n], Y[n]]^t \|}{\| [X[0], Y[0]]^t \|} = \prod_{i=0}^{n-1} \sqrt{1 + \sigma_i^2 2^{-2i}}, \quad (7)$$

where  $\|\cdot\|$  stands for the norm of the vector. Note that  $K$  is constant for the non-redundant scheme since  $\sigma_i$  is in  $\{-1, 1\}$ .

**Redundant :** Non-redundant CORDIC is slow inherently with delay of  $O(n^2)$  due to its recursiveness and serial dependency, since a micro-rotation with delay  $O(n)$  should be finished before processing the next micro-rotation. Delay performance of a macro-rotation ( $n$  micro-rotations) can be improved from  $O(n^2)$  to  $O(n)$  by using redundant arithmetic (carry-free addition such as carry save or signed-digit addition) to determine the direction of the rotation  $\hat{\sigma}_i$ , based on an estimate instead of an exact value [14]. The redundant arithmetic gives a delay of  $O(1)$  instead of  $O(n)$ , and the estimation of direction is necessary not to erode the advantage of  $O(1)$ . This requires the modification of the recurrences and selection function. This redundant CORDIC scheme produces the output about 4 times faster than the non-redundant [14]. However, it introduces additional cost since the scale factor  $K$  is variable depending on a macro-angle by allowing  $\hat{\sigma}_i$  to be in  $\{-1, 0, 1\}$ .

**Constant-Factor-Redundant :** To reduce implementation cost of redundant CORDIC, it would be good to have a constant scale factor by forcing  $\hat{\sigma}_i$  in  $\{-1, 1\}$ . However, since  $\hat{\sigma}_i$  is determined from an estimate, there arises a convergence assurance question. A scheme appending correcting iteration stages at proper positions was proposed for it [15]. Along to this idea, the number of extra correcting iterations is further reduced by dividing the micro-iterations (for  $i = 0$  to  $i = n - 1$ ) into two groups: one group where the direction of the rotation is in  $\{-1, 1\}$  for  $i = 0$  to  $i = n/2$  and the other in  $\{-1, 0, 1\}$  for  $i = (n + 1)/2$  to  $i = n - 1$  correcting iterations by 50 % since correcting iteration is not needed for the second half of the micro-iterations and we still obtain a constant scale factor  $K$  since the value of  $K$  in  $n$ -bit precision does not depend on the  $\hat{\sigma}$  value for  $(n + 1)/2 \leq i \leq (n - 1)$ .  $Z$ -recurrence also can be modified so that  $\hat{\sigma}_i$  is determined quickly by looking at a few most significant bits. This new scheme is called Constant-Factor-Redundant-CORDIC(CFR-CORDIC). The modified recurrences and selection functions for the scheme are described below.

$$\begin{aligned} X[i + 1] &= X[i] + \hat{\sigma}_i 2^{-i} Y[i] \\ Y[i + 1] &= Y[i] - \hat{\sigma}_i 2^{-i} X[i] \\ U[i + 1] &= 2(U[i] - \hat{\sigma}_i 2^i \tan^{-1} 2^{-i}) \end{aligned} \quad (8)$$

where  $U[i]$  is for the implementation simplicity, which is equal to  $2^i Z[i]$ , and the selection function is given as follows:

$$\hat{\sigma}_i = \begin{cases} 1 & \text{if } \hat{U}[i] > 0 \\ & \text{or } \hat{U}[i] = 0 \cap i < n/2 \\ 0 & \hat{U}[i] = 0 \cap i \geq n/2 \\ -1 & \text{if } \hat{U}[i] < 0 \end{cases} \quad (9)$$

When  $t$  fractional bits are used in the estimate value, i.e.,  $\hat{U}[i]$  is computed using  $t$  fractional bits of redundant representation of  $U[i]$ , the following correcting iteration need to be included, where the interval between indexes of correcting iterations should be less than or equal to  $(t - 1)$  up to the last iteration index equal to  $n/2$ . When the correction stage is necessary at the  $j$ th step of micro-iteration,

$$U^C[j + 1] = U[j + 1] - 2\hat{\sigma}_j^C 2^j \tan^{-1} 2^{-j} \quad (10)$$

with the direction of the rotation  $\hat{\sigma}_j^C$  determined from the same selection function of eq.( 9), except being decided based on  $\hat{U}[j + 1]$  instead of  $\hat{U}[i]$ .

### 3.2 Vectoring case

While the rotating case affords vector-wise rotation to implement a geometrical mapper, the vectoring case does elementary functions as in Table 1. Apparent difference between the vectoring and rotating mode is the zero enforcing parameter, which necessitates a different selection function. For the conventional CORDIC, the recurrence equations are given:

$$\begin{aligned} X[i + 1] &= X[i] + \sigma_i 2^{-i} Y[i] \\ Y[i + 1] &= Y[i] - \sigma_i 2^{-i} X[i] \\ Z[i + 1] &= Z[i] + \sigma_i \tan^{-1} 2^{-i} \end{aligned} \quad (11)$$

with the following selection function,

$$\sigma_i = \begin{cases} 1 & \text{if } Y[i] \geq 0 \\ -1 & \text{if } Y[i] < 0 \end{cases} \quad (12)$$

The selection function for CFR-CORDIC in vectoring has been developed shown below: Let  $W[i] = 2^i Y[i]$  in the same token as for the rotating case, then

$$\begin{aligned} X[i + 1] &= X[i] + \hat{\sigma}_i 2^{-i} Y[i] \\ W[i + 1] &= 2(W[i] - \hat{\sigma}_i X[i]) \\ Z[i + 1] &= Z[i] + \sigma_i \tan^{-1} 2^{-i} \end{aligned} \quad (13)$$

$$\hat{\sigma}_i = \begin{cases} 1 & \text{if } \hat{W}[i] > 0 \\ & \text{or } \hat{W}[i] = 0 \cap i < n/2 \\ 0 & \hat{W}[i] = 0 \cap i \geq n/2 \\ -1 & \text{if } \hat{W}[i] < 0 \end{cases} \quad (14)$$

Here the correcting stage at the  $j$ th step is defined as follows:

$$W^C[j + 1] = W[j + 1] - 2\hat{\sigma}_j^C X[j + 1] \quad (15)$$

So far, we discussed about recursive structures of several CORDIC schemes to implement the basic PE. The PE, augmented by a translator, necessitates scaling operation at each stage, because shuffling of the output at each stage makes continuous accumulation of the scaling factor complex to be processed at the final stage. The scaling operation has been solved either by an explicit way or an implicit. The explicit way is dividing the rotated vector by a constant, which is known for the non-redundant, to be calculated while running the micro-steps of CORDIC [12,14]. The division can be processed by another CORDIC (in a linear mode) or a divider. The implicit approach reconfigures the sequence of micro-iterations of the CORDIC, eventually to have a different norm from that without



scaling micro-iterations. Scaling micro-iterations target in general at making the adjusted scaling factor in a form of  $2^i$  or 1, which can be easily set to the unit size. Each micro-iteration can be composed of i) reduction axis-scaling [16], ii) repetition of vector-scaling, iii) expansion axis-scaling or combinations thereof. Relevant issues regarding search for the solution are to be further studied, better than the greedy method or the decomposed search [18]. In summary, the explicit scaling almost doubles the system complexity, while the implicit increases 25 % for non-redundant CORDIC and about 30 % for redundant CORDIC.

### 3.3 VLSI Scheme

To maximize the throughput of the geometric processor, the fully spanned architecture is selected. Affine transformer is a trivial case, which can be implemented by using a single CORDIC of which micro-iteration is expanded to include an addition. To implement a spherical transformer, 4 CORDICs are configured: i) circular square root of  $\sqrt{x^2 + y^2}$ , ii) hyperbolic square root of  $\sqrt{r^2 - (\sqrt{x^2 + y^2})^2}$ , and two iii) linear divisions of  $u$  and  $v$ . To get first estimates of the VLSI size, a typical TV image processing application is considered:  $O(10^5)$  pixel/image addressing and  $O(10^{-1})$ sec screen flashing. For the case, the number of input bits  $b_i \approx \sqrt{\text{pixel number}}$ , for which 12 bits are sufficient. To allow possible interpolations between pixels,  $b_f$  is set to be 16. Each CORDIC module requires  $(b_i + \log_2 b_i)$  steps of micro-iterations, and 30% additional iterations for an implicit scaling.

For the spherical transformer, using fully spanned 4-CORDIC, the number of TRs are estimated about 30K ( $4 \times 6K \times 1.3$ ).

## References

- [1] R. Nicholl and T. Nicholl, "Performing Geometric Transformations by program Transformation," *ACM Trans. on Graphics*, Vol. 9, No 1, pp.28-40, 1990.
- [2] N. Ansari and E. Delp, "Recognizing Planar Objects in 3-D Space," *Proc. of SPIE*, Vol. 1197, pp.127-138, 1989.
- [3] R. Cossu, M. Ercoli and L. Moltedo, "Extension of CGI functions for Generation and Manipulation of Raster Image," *Computers & Graphics*, Vol.13, No 1, pp.39-48, 1989.
- [4] T. Nakanishi and H. Yoshimura, "A High-speed Address Generator for Affine Transformation," *Nat. Conv. IECE*, 1985.
- [5] ACM/SIGGRAPH Graphics Standards Planning Committee, Report of the CORE Definition Subgroup, 1977.
- [6] H. Yoshimura, T. Nakanishi and H. Yamauchi, "A 50-MHz CMOS Geometrical Mapping Processor", *IEEE Transactions on circuits and systems*, Vol 36, No. 10, pp.1360-1364, October 1989

- [7] K. Arbter and et.al., "Application of Affine-invariant Fourier Descriptors to Recognition of 3-D Objects," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 12, No 7, pp.640-647, July 1990.
- [8] T. Wakahara, "On-line Handwritten Character Recognition Using Local Affine Transformation," *Systems and Computers in Japan*, Vol.20, No 7, pp.10-19, July 1989.
- [9] K. Aono, M. Toyokura and T. Araki, "30nsec (600 Mops) Image Processor with a Reconfigurable Pipeline Architecture," *Proc. IEEE 1989 Custom Integrated Circuits*, pp.24.4.1-4, 1989.
- [10] E. Maxwell, *General Homogeneous Coordinates in Space of Three Dimensions*, The University Press, Cambridge, England, 1961.
- [11] J. Eldon, Z. Stroll and E. Swartzlander, "Image Processing Address Generator Chip," *Proceedings of IEEE Int. Conf. Acoustics, Speech, and Signal Processing 3*, pp.993-996, 1985.
- [12] J. Walther, "A Unified Algorithm for Elementary Functions", *AFIPS Spring Joint Computer Conference*, pp.379-385, 1971.
- [13] H. Kung, "Let's Design Algorithms for VLSI Systems," *Caltech Conf. VLSI*, pp.65-90. 1979.
- [14] M. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD", *IEEE Trans. on Computers*, Vol. C-39, No 6, pp.725-740, June 1990.
- [15] N. Takagi, T. Asada and S. Yajima, "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation", Submitted to *IEEE Trans. on Computers*, 1989.
- [16] G. Haviland and A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. on Computers*, Vol C-29, No 2, pp.68-79, Feb. 1980.
- [17] J. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces", *Proceedings of IEEE Int. Conf. Acoustics, Speech, and Signal Processing 2*, pp.927-930, 1983.
- [18] J. Lee and T. Lang, "Matrix Triangularization by Fixed-point Redundant CORDIC with a Constant Scale Factor", *Proc. SPIE Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations*, July 1990.
- [19] S. Note et. al., "Automated Synthesis of a High Speed CORDIC Algorithm with the CATHEDRAL-III Compilation System", *Int. Conf. Circuit and System*, pp.581-584, 1988.

# Performance of Defect-Tolerant Set-Associative Cache Memories<sup>1</sup>

J. F. Frenzel

Department of Electrical Engineering

University of Idaho, Moscow, Idaho 83843

jffrenzel@groucho.mrc.uidaho.edu, 208-885-7888

**Abstract-** Increased use of on-chip cache memories has led researchers to investigate their performance in the presence of manufacturing defects. Several techniques for yield improvement are discussed and results are presented which indicate that set-associativity may be used to provide defect -tolerance as well as improve the cache performance. Tradeoffs between several cache organizations and replacement strategies are investigated and it is shown that token-based replacement may be a suitable alternative to the widely-used LRU strategy.

## 1 Introduction

The dramatic increase in cache memory size and diminishing geometries has resulted in lower yields. Today's high performance processors often have on-chip cache and consequently the yield of these memories can be a significant factor in determining the ultimate cost of the processor. One way of increasing yields is to provide defect-tolerance through the use of redundant resources. Two methods for achieving defect-tolerance are commonly employed in the design of dynamic RAM (DRAM) memories, namely the use of error correcting codes and spare rows and columns [5]. However, both techniques result in increased circuitry and possible increases in access times.

Associative memories offer an alternative approach. By design, associative memories have the flexibility necessary to function in the presence of defects. With the inclusion of control logic it is possible to force the memory to operate "around" the defect and use alternative locations, albeit with a reduction in storage capacity. In the following sections we will describe basic cache memory operation and then discuss the different techniques for providing defect-tolerance.

## 2 Cache Operation

A cache memory is a fast intermediary memory positioned between a processor and main storage. The goal of a hierarchical memory system is an average access time close to that of the cache memory, at a cost per bit approaching that of the main memory. To achieve the former the cache must be designed to keep the most frequently referenced items in the cache. A system may be designed with separate caches for data and instructions or a single (unified) cache.

---

<sup>1</sup>This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

## 3.2.2

## 2.1 Organization

A cache memory is organized as sets of blocks, where each block is typically 4 to 16 bytes of data from main storage. In a direct-mapped cache each set consists of only one block, whereas in a  $n$ -way, set-associative (SA) cache each set contains  $n$  blocks. The total cache size is the product of the block size, the number of sets, and the associativity,  $n$ .

## 2.2 Address Translation

Address references to the cache are split into three fields, the widths of which depend upon the cache size and organization. The block field is used to index a particular item within a block and is  $\log_2 b$  bits wide, where  $b$  is the number of addressable items within a block. If  $s$  is the number of sets in the cache, then the set field is  $\log_2 s$  bits wide and is used to indicate a particular set for access. The remaining bits are referred to as the *tag* and are used to distinguish between other blocks of main memory which may be stored in the same set. Each block in a set has storage to hold both the block data and the tag associated with that block. The collection of tag storage for the cache is referred to as the *tag directory*. During a memory access, the tag field for the address is compared with all entries in the tag directory corresponding to the referenced set. If there is a match, the data from the matched block is sent to the processor. If there is no match, referred to as a miss, then the missed data must be loaded from main memory.

## 2.3 Replacement Policy

On a miss, the cache must decide where to place the block from main storage which caused the miss. For a direct-mapped cache the decision is trivial, as each block from main storage maps to a single block in the cache. However, with a set-associative cache, assuming the referenced set is full, there are  $n$  possible blocks to replace. One of the best replacement algorithms is referred to as *least recently used* (LRU), where the set is treated as a stack and accessing a particular block moves that block to the top of the stack. The least recently used block is always at the bottom of the stack and a miss will load the data into this block and move it to the top of the stack. Efficient implementations of the LRU replacement algorithm require  $n(n-1)/2$  bits of storage per set to maintain the  $n!$  possible stack configurations. Consequently, a 4-way SA cache requires 6 bits of storage per set, while an 8-way SA cache requires 28 bits per set. Additional circuitry is needed to update the stack configuration as a result of an access. Alternative replacement strategies are first in, first out (FIFO), and random. The FIFO algorithm is implemented using a modulo  $b$  counter for each set, incremented on every miss to that set. One technique for implementing a pseudo-random replacement strategy is to use a single modulo  $b$  counter for the entire cache and increment it on every miss, regardless of the set. This will be referred to as token-based replacement.

## 2.4 Discussion

Several observations may be made in comparing direct-mapped caches to set-associative caches. First, for a given cache size, the tag field and subsequently the tag directory will be larger for the SA cache. This is because the  $n$ -way, set-associative cache will have  $1/n$  the number of sets as the direct-mapped cache, needing fewer bits in the set field, and increasing the number of bits in the tag field. Second, for the set-associative cache,  $n$  comparisons must be conducted in parallel between the tag field and the entries in the tag directory. Furthermore, the set-associative cache has an additional delay over the direct-mapped cache as a result of the need to multiplex the data from each of the blocks in the referenced set to the output. Lastly, the SA cache has additional circuitry needed to implement the replacement algorithm.

## 3 Defect-Tolerance Strategies

### 3.1 Spare Resources

There are several methods for implementing memory reconfiguration in the presence of defects: electrically programmable links, electron-beam programmable fuses, and laser cutting/welding [6]. These techniques can be employed to bypass faulty resources and activate spare units. The most common technique for increasing memory yields is to include spare rows and/or columns in the data array and sufficient programmable decoders. While all implementations increase the circuit area, some methods may also increase the access times and power dissipation [5]. Furthermore, unless special circuitry is added it is usually not possible to test the spare rows/columns without first programming the decoders.

It has recently been observed that manufacturing "throughput", measured in usable chips per unit time, is dominated by the delay associated with repairing defective parts rather than the process yield [2]. These researchers argue that efforts should be directed at maximizing the throughput, rather than the yield, and propose algorithms for achieving this by balancing repair time and yield of repaired parts. Previously, production experience with a 64K DRAM indicated that the repair algorithm typically took several seconds and represented roughly half of the entire test time [10]. The next two sections describe methods which eliminate the time needed to execute a repair algorithm.

### 3.2 Error Correcting Codes

Error correcting codes can be used to correct single or multiple errors in the tag directory and data array caused by manufacturing defects. Codes may be selected to provide a guaranteed level of protection at a corresponding increase in circuit area and access time. A 16-bit word would require 6 extra check bits to *detect* and *correct* all single errors.

In addition to storing the check bits, additional circuitry is needed to encode or decode during memory accesses. For large words, where the use of check bits is most efficient, the delay associated with this circuitry can be significant. Results of a timing analysis are

### 3.2.4

presented in [11] which indicate a 20% increase in access time using single error correction, double error detection coding of the tag directory and data array. For these reasons, error correcting coding is generally reserved for applications which require tolerance of transient errors incurred during normal operation. However, Mostek built a 1-Mbit ROM with a 32-bit word that achieved a 3-fold improvement in yield at a 20% increase in area [8].

A distinct advantage of error correcting coding is the lack of any "repair" time. As mentioned earlier, the delay associated with this process can severely affect the manufacturing throughput for the part.

## 3.3 Associativity

Sohi observed that a cache memory does not have to be defect free to meet its objective, namely reduce the average memory access time of a hierarchical memory system [11]. A direct-mapped cache memory with a defective block will never be able to hold items from main memory which map to that set in the cache. For a cache to operate properly under this condition two things are necessary: one, the cache must be able to recognize a defective block and generate a miss and two, must have the capability of performing a load through, so that the processor can access the item. An associative cache has alternate locations within a set which can be used when there is a defective block present. Ideally, the circuitry which implements the replacement algorithm would be modified at test time to exclude defective blocks from selection during replacement. Provided each set has at least one good block all items from main memory can map to a good location in the cache.

## 4 Related Work

Patterson et al. described the implementation of a cache memory in which each cache block was provided with a *fault tolerant* bit, which could permanently invalidate a cache block. Set-associativity was achieved through the use of multiple chips and block replacement was directed by a token [7]. Accessing a bad block would result in a miss.

More recently, Bergh et al. designed a fully associative fault-tolerant memory. Extra logic, amounting to a 2% increase in area, allowed the memory to completely bypass defective locations transparent to the user [1].

Finally, Sohi investigated the performance under defects, as measured by miss ratio, of three different cache organizations: direct-mapped, 2-way set-associative, and fully-associative [11]. His research illustrated that it is possible for a 2-way set-associative cache, using a LRU replacement strategy, to outperform a direct-mapped cache of equivalent size in the presence of defects.

This paper attempts to extend the work of Sohi in evaluating the benefits of associativity for the purpose of defect-tolerance. In this paper I focus upon set-associative cache memories for the following reasons:

- fully-associative caches are generally not required for many applications and are prohibitively expensive;

% Change in Hits Compared to 2K, DM Cache							
Size (words)	DM	2-way, SA			4-way, SA		
		LRU	FIFO	Token	LRU	FIFO	Token
2K	0	2.2	1.7	1.8	3.5	2.8	2.9
4K	5.0	7.6	7.2	7.2	8.8	8.2	8.2
8K	9.6	12.0	11.7	11.8	13.1	12.6	12.7

Table 1: Performance of Defect-free Caches

- set-associative caches possess the flexibility necessary to reduce the impact of defects.

Specifically, this paper investigates set-associative caches of various organizations under three different replacement strategies, least recently used (LRU), first-in, first-out (FIFO) and token-based. The LRU strategy is widely accepted as the superior strategy, although costlier to implement [9].

## 5 Simulation Methods

Performance evaluation was conducted using address trace simulation. The address traces were generated from runs of SPICE, gcc, and TeX, for a total of over 2.8 million references, approximately 75% of which were instruction references [3]. All address references were assumed to reference items of the same size, namely one word. A wide variety of caches were studied; however, in all cases the block size was held at 8 words and the cache was treated as a unified cache (instructions and data).

Three different cache sizes were simulated, ranging in size from 2K words to 8K words. For each size, three different cache structures were investigated: direct-map, 2-way set-associative, and 4-way set-associative. Each associative cache was simulated using three different replacement strategies: least recently used (LRU), first in, first out (FIFO), and token-based. Lastly, each associative cache was simulated under three different levels of defects, ranging from zero to 25%.

During defect simulation each cache was simulated forty times, each iteration using a random distribution of defects. Furthermore, defect-levels were limited and the defects distributed such that each set was guaranteed to have at least one good block. The replacement strategies were modified from the traditional descriptions to prevent loading a missed block into a defective location.

## 6 Results

### 6.1 Defect-free Performance

Table 1 shows the percent change in the total number of hits for various cache organizations, relative to the total number of hits for a 2K, direct-mapped (DM) cache. From this data we can make several observations regarding the relative performances of various organizations under defect-free operation:

Size (words)	% Change in Hits Compared to 2K, DM Cache					
	2-way, SA			4-way, SA		
	LRU	FIFO	Token	LRU	FIFO	Token
2K	-1.0	-1.4	-1.6	1.9	1.2	1.3
4K	5.7	5.4	5.2	7.5	6.9	6.8
8K	10.3	10.1	9.9	12.0	11.4	11.5

Table 2: Performance with 12.5% Defect-Level

- As cache size doubled there was approximately a 5% increase in hits, relative to a 2K, DM cache, across all structures and replacement algorithms. However, this effect would eventually diminish as the cache size approached that of the workload's working set.
- The token-based replacement algorithm was virtually identical in performance to the FIFO algorithm for all cache organizations. While at first this may seem surprising, neither algorithm is a "usage based" algorithm and consequently their performance is roughly equivalent.
- LRU was the best replacement strategy, increasing the performance by roughly 0.5% over the other algorithms. For a fixed cache size, the performance difference increased with associativity. As the number of blocks per set increased, LRU's superior management of those resources became more apparent. For a fixed associativity,  $n$ , the improvement decreased with increasing cache size. This may be attributed to reduced contention in the cache.
- Doubling the associativity increased performance by approximately 2%, with the improvement diminishing as the associativity increased. Again, this may be attributed to reduced contention within the cache.

As cache size increases, there is less contention for space in the cache and performance differences due to associativity and replacement strategies tend to diminish. The same is true for a fixed cache size as associativity increases, particularly under LRU replacement.

## 6.2 Performance under Defects

Tables 2 and 3 detail the results of simulating various cache organizations under two different defect-levels. As in the first table, the numbers represent the percent change in the total number of hits, relative to a defect-free, 2K, DM cache. At the 12.5% defect-level, there was a drop in performance that was a function of both cache size and associativity, but *not* replacement strategy. For example, all 2K, 4-way, SA caches experienced a decline of approximately 1.6% from their defect-free performance. This can be attributed to the fact that each replacement algorithm was modified such that missed data would never be loaded into a defective block. The average number of bad blocks per set is equal to the product of the associativity and the defect-level. So at a 12.5% defect-level a 2-way cache



% Change in Hits Compared to 2K, DM Cache						
Size (words)	2-way, SA			4-way, SA		
	LRU	FIFO	Token	LRU	FIFO	Token
2K	-4.1	-4.4	-4.7	0.2	-0.5	-0.4
4K	3.8	3.6	3.4	6.0	5.4	5.3
8K	8.5	8.4	8.0	10.7	10.2	10.1

Table 3: Performance with 25% Defect-Level

will have, on average, 0.25 bad blocks per set, or one bad block for every four sets, while a 4-way cache will average one bad block for every other line. Consequently, the effect of defects is to decrease the associativity. At low defect-levels, and particularly for low values of associativity, the decrease will be minor and thus the performance differences between replacement algorithms will remain approximately constant.

In general, the larger the associativity or the total cache size, the smaller the drop in performance due to defects. Increasing associativity or size are two methods for reducing contention in a cache and consequently it is expected that defects would have a lesser effect on these caches. Another important observation, is that all 4-way, SA, 2K caches, regardless of replacement algorithm, outperformed the defect-free, DM, 2K cache. Furthermore, for caches larger than 2K, all associative caches with a 12.5% defect-level outperformed a defect free DM cache of equivalent size. This is a clear example of using associativity to provide defect-tolerance *and* a performance improvement. At a defect-level of 25%, only the 4-way, set-associative caches outperformed the defect-free, DM caches.

Other researchers have suggested that the use of associative cache memory may be on the decline because as cache memories increase in size the performance difference between direct-mapped and set-associative will decrease [4]. Furthermore, a DM cache is always smaller and faster than a SA cache of equivalent capacity, due to the extra circuitry required to implement the associativity. From our limited trials it is difficult to validate such a trend in performance. An 8K, DM cache had 9.6% more hits than a 2K, DM cache, whereas the 2-way, SA cache had 12% more and the 4-way had 13% more. These differences are similar to the differences observed for 2K caches. Of course, common sense dictates that as the cache size approaches the size of the working set the differences will diminish. While this may occur soon for board level cache memories, the author suspects that on-chip cache will continue to benefit from the use of associativity, due to size limitations. Doubling the associativity and halving the number of sets requires less area than doubling the cache capacity.

## 7 Summary

The results indicate that a set-associative cache can experience a significant number of defects and still exceed the performance of a direct-mapped cache of equivalent capacity. Secondly, although the LRU replacement strategy performed better than FIFO or token-based replacement, the modest improvement, particularly at lower associativities and large

cache sizes, may not warrant the increase in control logic.

The fundamental question is: "Should associativity be used to increase manufacturing yields instead of spare rows and columns?" To answer this, one needs to develop a cost function capable of reflecting the impact of manufacturing throughput, circuit characteristics (power, size, speed), and cache performance as measured by miss ratio. Several observations may be made:

- If the cache access time is critical and the application can not tolerate the additional delay imposed by associativity then spare rows and columns are the only alternative for increasing yields.
- If the chosen technology has matured to the point where manufacturing throughput is not severely affected by the time needed to repair devices, then spare rows and columns are probably the logical selection. A repaired part will be guaranteed to have a full set of defect-free blocks and will have known performance characteristics.
- If, on the other hand, manufacturing throughput is poor, due either to low yields or lengthy repair times, then using associativity may be viable alternative to using spare rows and columns. By doubling the associativity and halving the number of sets, cache performance can be improved even in the presence of defective blocks. Repair time will be minimal and simply involve marking defective blocks as unusable. Research is being considered to evaluate the area overhead associated with enhancing the replacement algorithms to avoid defective blocks.

Perhaps the biggest deterrent to using this approach may be the difficulty in marketing such a device. Customers expect devices to be 100% defect-free and might be unwilling to order parts which are guaranteed to have "a maximum defect-level," particularly as two devices with the same defect-level will not perform identically on the same workload.

## References

- [1] Harald Bergh et al, "A fault-tolerant associative memory with high-speed operation," *IEEE Journal of Solid-State Circuits*, pages 912 - 919, August 1990.
- [2] Ramsey W. Haddad et al, "Increased throughput for the testing and repair of RAM's with redundancy," *IEEE Transactions on Computers*, pages 154 - 166, February 1991.
- [3] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.
- [4] Mark D. Hill, "A case for direct-mapped caches," *IEEE Computer*, pages 25 - 40, December 1988.
- [5] Will R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield," *Proceedings of the IEEE*, pages 684 - 698, May 1986.

- [6] R. Negrini et al, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, chapter 4, The MIT Press, 1989.
- [7] David A. Patterson et al, " Architecture of a VLSI instruction cache for a RISC, " In *Proceedings of the Tenth Annual Symposium on Computer Architecture*, pages 108 - 116, June 1983.
- [8] T. Shinoda et al, " A 1Mb ROM with on-chip EEC for yield enhancement," In *IEEE International Solid State Circuits Conference*, pages 158 - 159, February 1982.
- [9] Alan Jay Smith, " Cache memories, " *ACM Computing Surveys*, pages 473 - 530, September 1982.
- [10] Robert T. Smith et al, " Laser programmable redundancy and yield improvement in a 64K DRAM," *IEEE CJournal of Solid-State Circuits*, pages 506 - 514, October 1981.
- [11] Gurindar S. Sohi, " Cache memory organization to enhance the yield of high-performance VLSI processors, " *IEEE Transactions on Computers*, pages 484 - 492, April 1989.

1. The first part of the document is a letter from the President of the United States to the Congress, dated January 1, 1861. It is a very important document, as it sets out the President's policy for the new year. The President states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

2. The second part of the document is a report from the Secretary of the Treasury, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

3. The third part of the document is a report from the Secretary of the Interior, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

4. The fourth part of the document is a report from the Secretary of the War, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

5. The fifth part of the document is a report from the Secretary of the Navy, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

6. The sixth part of the document is a report from the Secretary of the State, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

7. The seventh part of the document is a report from the Secretary of the War, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

8. The eighth part of the document is a report from the Secretary of the Navy, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

9. The ninth part of the document is a report from the Secretary of the State, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

10. The tenth part of the document is a report from the Secretary of the War, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

11. The eleventh part of the document is a report from the Secretary of the Navy, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

12. The twelfth part of the document is a report from the Secretary of the State, dated January 1, 1861. It is a very important document, as it sets out the Secretary's policy for the new year. The Secretary states that he is pleased to see the Congress assembled, and that he is confident that the country is in a good position to meet the challenges of the future.

## S P R O C — A Multiple-Processor DSP IC

R. Davis  
Hewlett-Packard ICBD  
Corvallis, OR

**Abstract-** A large single-chip multiple-processor digital signal processing IC fabricated in HP-Cmos34 is presented. The innovative architecture is best suited for analog and real-time systems characterized by both parallel signal data flows and concurrent logic processing. The IC is supported by a powerful development system that transforms graphical signal flow graphs into production-ready systems in minutes. Automatic compiler partitioning of tasks among four on-chip processors gives the IC the signal processing power of several conventional DSP chips.

## 1 Introduction

Digital signal processing (DSP) involves the real-time acquisition of analog (continuous) inputs, their analysis and processing in a digital system, and subsequent synthesis and reintroduction back to the analog domain.

Conventional DSP chips are tuned for fast multiply and multiply-and-accumulate (MAC) algorithms on serial data streams such as required for filtering and spectral analysis. These algorithms take the ubiquitous form

$$y(n) = \sum_{i=1}^{N-1} a(i) * x(n-i) + \sum_{k=1}^M b(k) * y(n-k)$$

that compute outputs as weighted sums of present and past inputs, and past outputs. However, many analog and real-time systems are better characterized by complex networks of parallel, and often asynchronous, data flows and concurrent logic processing. Programming a conventional DSP chip to perform fundamental scheduling and synchronization tasks can become intractable.

**SPROC**<sup>1</sup>, an IC and development system, efficiently manages concurrency through the use of dedicated control circuitry and a powerful compiler that automatically and transparently partitions tasks among several processors. It minimizes the number of components for simple systems, yet remains largely extensible for arbitrarily complex designs; it is easier to program with its library of customizable building blocks; it is easier to debug with its built-in real-time probe; it facilitates both rapid prototyping and production development on one system. It features full 24-bit fixed-point precision with 56-bit accumulation resulting in a 144dB dynamic range for signal bandwidths up to 250 kHz and handles all signal scaling automatically. The chip can be dynamically reprogrammed,

<sup>1</sup>SPROC is the registered trademark of Star Semiconductor of Warren, NJ (908) 647-9400.

making adaptive, self-calibrating, and field upgradeable systems easier to design. The parallel port supports Motorola and Intel microprocessor interface protocols. The IC can be ganged to implement arbitrarily complex systems.

## 2 Chip Programming and Development Cycle

First, a signal-flow diagram of the desired system is graphically captured by selecting, placing, interconnecting, and parameterizing standard or customized function blocks, such as signal generators, summers, filters, etc. Next, the compiler converts the signal-flow diagram into executable code, allocating tasks efficiently between the available processors, building symbol tables for simple interfacing to the code. Then, the code is downloaded to the **SPROC** chip via either the development or target system. Finally, while the code is executing, circuit nodes can be probed, parameters can be modified, and the system observed in real time.

The **SPROC** advantages are fundamental: more complex, analog and real-time applications can be realized in a fraction of the time; designs can be observed in real-time and modified on-the-fly; any design that can be compiled is guaranteed to run on the **SPROC** chip. Higher designer productivity and improved performance translates into short time-to-market of more creative and competitive systems.

## 3 Chip Architecture

A *Harvard* architecture employing separate program and data busses allows concurrency in instruction fetch, decode, execution and data manipulation. The major blocks are the general signal processor (**GSP**), parallel interface (**HOST**), a serial interface (**ACCESS**), serial interfaces for sampled data (serial **PORTS**), a **DAC** port, a glue block (**GLUE**), and memory. An overview of the system architecture is shown in Figure 1.

**SPROC** operates in various configurations and modes. In *Master* mode, the system boots from external *EPROM*. In *Slave* mode, **SPROC** responds to an external controller which is either a microprocessor or a master **SPROC**. In *Redundancy* mode, the **GSPs** perform a system self-test, attempts redundancy and reconfigures the system. Thus, while the chip is highly integrated, it is flexible and extensible.

### 3.1 GSP

Each **GSP** is a 24-bit digital processor with 64 instructions and eight addressing modes. Main blocks include program control, address generator, multiplier, **ALU**, and decoder. Instructions include multiply (**MPY**) and multiply-and-accumulate (**MAC**) that execute in fifteen clock periods. One of up to four **GSPs** control both program and memory busses on a time-multiplexed basis. As triggered, a time slice for I/O operations via **HOST**, **ACCESS**, **PORTS**, or probing **DAC** is interjected. (see Figure 2)

$\bar{P}$  = Program Bus Access

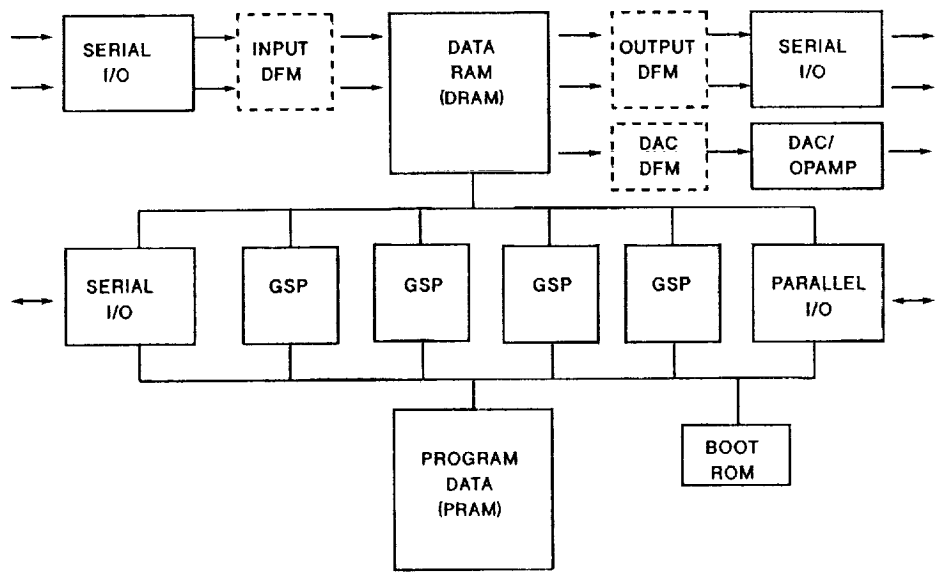


Figure 1: System Architecture

	cycle 0	cycle 1	cycle 2	cycle 3	cycle 4
GSP1	P		D		
GSP2		P		D	
GSP3			P		D
GSP4	D			P	
I/O		D			P

Figure 2: System Timing

### 3.3.4

D = Data Bus Access

I/O = HOST, ACCESS, PORTS, or probing DAC Access

In *Redundancy* mode, each **GSP** executes a self-test code from internal ROM upon power-up. If defective, the **GSP** is essentially held in reset and removed from tasking operations. This enables otherwise functional parts to yield at wafer test and provide fault-tolerance in the field. The fault coverage of this test is approximately 70%.

## 3.2 HOST

The host interface (**HOST**) is a 24-bit asynchronous bidirectional parallel port with a 64K addressing range, and supports 8, 16, and 24 bit transfers. It typically interfaces to the digital subsystem of the target environment. The **GSPs** can access the **HOST** via **LOAD** and **STORE** instructions. Internally, **SPROC** has a 12-bit addressing range with 4 bits reserved for master to slave addressing for memory-mapped devices or ganged **SPROCs**.

## 3.3 ACCESS

The access port (**ACCESS**) is a two port serial interface. It is typically used to observe and modify the contents of internal memory while the system is operating. The input port requires data, clock, and strobe; the output port drives a strobe and data based on the input port clock rate. Access is time multiplexed and is transparent to internal operations. Full read/write access is provided to any valid **SPROC** address.

## 3.4 PORTS

The sampled data streams are supported by four serial ports configurable for data, clock, strobe, and sync. There are two input and two output ports available. A data flow manager (**DFM**) manages the concurrency of multiple **GSP** and data RAM accesses. Very simply, an input **DFM** writes input sample data to consecutive data RAM locations and updates a write pointer. An output **DFM** will subsequently fetch output sample data from the data RAM.

## 3.5 GLUE

The glue block (**GLUE**) provides address decoding and memory mapping, mode control, system cycle generation, and serial port timing.

## 3.6 DAC

The digital-to-analog port (**DAC**) allows the probing of any node on the signal-flow diagram. These nodes are represented internally as two's complement FIFO buffers in data RAM. Hence, a node can be selected to direct its data buffer to the on-chip **DAC** port, and the analog value can be observed in real-time. An internal gain register can be loaded



to scale the digital value before outputting. The corresponding analog voltage is buffered and driven off chip, and may be observed with an oscilloscope, spectrum analyzer, etc.

## 4 IC Design Methodology

### 4.1 Partitioning

Star Semiconductor approached HP with a prototype system breadboarded with off-the-shelf memory and Xilinx and Actel field-programmable gate arrayed logic and a desire for fast, integrated silicon. Chip development on the customer side was primarily in *Cadence*; with VERILOG providing functional, behavioral, and logic simulation of the system and VERIFault for fault analysis. TA, a static timing analyzer was used for detailed timing optimization.

HP recommended developing additional standard cells including a recirculating flip-flop, adder, and lookahead cells to complement its standard cell offering HP-Cmos34. This resulted in enhanced performance, less silicon area, and a more direct mapping of the netlist. We also developed the memories, DAC, and OSC and the task of global composition and verification. Critical paths were simulated in SPICE, and capacitance was fed back to the customer for final timing simulations. Clock, power, and analog routing required manual editing.

### 4.2 New Standard Cell Development

Realizing the prevalent use of recirculating registers led to the incorporation of 2, 3, and 4-way multiplexers into the flip-flop to minimize area. (See table 1)

Table 1: Comparison of flip-flops, multiplexer combinations

	Library	Width uM	Intrinsic Delay nS	Load Multiplier nS/pF
DFFB	Standard	54.6	7.8	3.4
DFFF	Standard	121.8	2.6	1.5
X1RG1	New-Std	46.2	1.9	2.1
MUX2B	Standard	37.8	2.9	4.8
XMUX2	New-Std	33.6	1.8	1.3
X2RG1	New-Std	71.4	1.9	2.2

### 3.3.6

Also, adder cells were developed including a slow 1 bit adder for the multiplier, a fast 4 bit adder, and a 4 bit carry lookahead for the address logic. (See table 2)

Table 2: Adder cells

	Library	Width uM	Intrinsic Delay nS	Load Multiplier nS/pF
XADD1B	New-Std	63.8	4.2	2.4
XADD4	New-Std	226.8	1.8	3.6
XLOOK4H	New-Std	189.0	1.6	2.9

## 5 Composition

A standard methodology of composing chips with multiple standard cell and custom blocks with the autorouting (HARP) tools has been developed. First, blocks are routed with random port locations to determine size. Then, blocks are re-routed with assigned port locations determined by the floorplan. Finally, the top level is routed with the pads. Developing the SPROC chip produced some enhancements to the process.

### 5.1 Routing Tricks

Initial block sizes were estimated using the `csize` program (which counts cells and adds their areas) with estimates for routing overhead. Port locations were assigned manually taking into account the initial floorplan and stored in a file for repeated runs and easy modification; random assignments were only made if a block had no assignment file. After iteratively routing to reach an optimal block size, a frame was extracted and placed in a dummy BDL file, which was then combined with custom frames for global routing including pads.

The new approach had the major advantage of flexibility of accepting new netlists from the designers and in experimenting with different partitions and floorplans in short order. Any piece could be easily rerouted and incorporated as desired, including the global route.

It was a must that each of the GSPs have optimal and identical performance, yet floorplan well. To accomplish this, ports were duplicated on each side of the block, and the blocks mirrored and routed back-to-back. To reduce the global routing, the block consisting of two GSPs only had one set of ports.

Routing ALLPORTS, INTERFACE, and GLUE as a single HARP block caused a great dispersal of the major busses. Partitioning these blocks and ports next to a central bussing channel proved to be more successful.

## 5.2 Routing Traps

Global power routing was problematic. Power estimates were determined by SPICE and the logic simulators. A package was selected to provide several power pads on each side. This required additional HARP modification. Also, end cap cells were modified to supply both power supplies to either end of the blocks, reducing IR drops by a factor of two. HARP was given parameters to increase the sizing of power busses between the blocks, each of which had multiple power ports. Manual editing was required to tie major power straps together, which run in pairs throughout the chip. The analog section was isolated by breaking the pad ring and connecting it to dedicated power pads. Also, digital signal lines were manually re-routed to avoid cross the analog logic.

Long global bussing of minimum width clock lines proved to have unaccepted RC wiring delays after final routing. The clock tree had to be resimulated taking these additional delays into account. To minimize skew, the clock drivers had been placed in the GLUE block, with the clock ports dispersed along one edge. The lines were selectively widened to a full contact width without penalty. It was sometimes possible to double the width of a single line if the vias on adjacent lines were coincident, or to drop the metal layers in parallel over long isolated runs. The clock network was reduced to a clock grid by effectively shorting the clock branches back together at the top level.

## 6 Custom Modules

### 6.1 RAM

The data and program memories are identical 1K word by 24-bit six-transistor static RAMs. A custom RAM was leveraged to improve the performance, as well as reduce area, with respect to an available RAM generator. The single-core array was developed for simplicity as 128 rows of 192 six-transistor static RAM columns. An 8-to-1 column multiplexer feeds a passive sense inverter and non-inverting tristate output buffer to achieve a 16ns cycle time in an area less than 10mm<sup>2</sup>. About 80 % of the area is consumed by the core array. A dual clocking mode for precharge was adopted. In half-cycle mode, the timing is determined by two edges of the system clock up to 40MHz. In internal clock mode, an inverter delay chain times the precharge against one edge of a clock up to 50MHz. (4.75V, 85°C) With a 20ns cycle boundary, the address generation gate delays, wiring delays, and clock skew must be less than 4ns for 50MHz operation. Both RAMS are accessed every clock cycle and consume approximately 600mW each.

### 6.2 ROM

The internal ROM is 512 words by 24 bits. The core is organized as 64 rows and 192 columns. The cycle time for the ROM is less than 16ns. (4.75V, 85°C) The ROM address space overlaps the program RAM; while the system is booting the program RAM data drivers are disabled. The ROM artwork was logic simulated to verify the bit programming.

The ROM area is 0.84mm<sup>2</sup>.

## 6.3 Analog Blocks

The **OSC** is an internal ring oscillator which minimizes component count for lower cost systems. The oscillator drives the system cycle generator when selected. An inverter feedback ring was chosen for simplicity. To reduce the frequency variability, the ring feedback is adjustable via programmable clocked-inverter taps decoded from three dedicated pins. The frequency variability is reduced to 36% over temperature and 17% over voltage over a tunable range of 30MHz to 80MHz. A schmitt trigger ring driver clocks a toggle flip-flop to insure a 50% duty cycle. In *Test* mode the oscillator is observable via a serial port. The oscillator resides in the pad ring to isolate it from the digital environment.

The **DAC** was selected from HP's customizable analog cell library available in HP-Cmos34. It is based on an 8-bit poly-resistor string design. Of note are Cmos transmissions gates used to make the resistor endpoints extendible to VDD and GND. The output swings between these voltage references which are sourced off-chip.

The **OPAMP** is a general purpose opamp that has a two-stage input and class AB output is used as a voltage follower to buffer the high-impedance **DAC** output. The opamp can swing rail-to-rail while driving a 3K resistive and/or 200pF capacitive load. An external compensation capacitor allows processing in Cmos34 without an extra mask required for linear capacitors.

## 7 Test Methodology

A 50MHz data rate speed goal made the Schlumberger S50 the local tester of choice. The customer contracted with TSSI (Beaverton,OR) for their software test development system (TDS) which converts captured simulation vectors to test vectors. TDS generates S50 MDC (patterns), TEG (timing), and pingroups directly. A pattern bridge (PBridge) essential samples the simulation responses, checking and formatting for S50 constraints. More than 900K vectors have been generated.

## 8 Results

First silicon was largely functional, with a major exception being the corruption of one of the processor addressing modes. Root cause was traced to a logic inversion in a Verilog model for a multiplexer. As a result, first silicon could not boot from ROM and hence run the redundancy code for self-test and configuration.

Second silicon was a quick, metal1/via/metal2 turn to correct the addressing mode, and the silicon was fully functional for software development and system operation up to 20MHz.

Third silicon was a full mask turn to increase the performance of the part. Unfortunately, a consequence of some of the edits introduced contention on the processor address

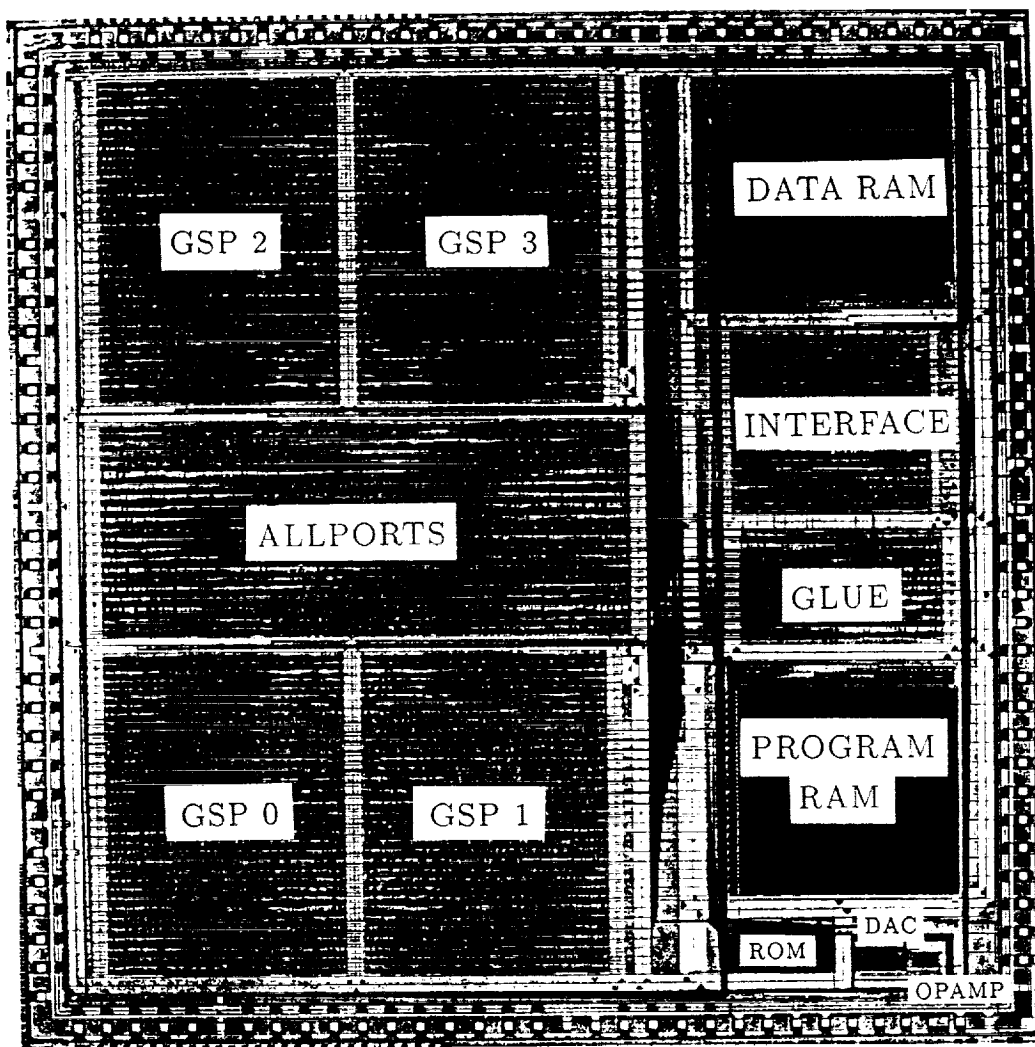
bus, limiting performance. Again, a quick turn is in the offing to solve the contention and improve the performance.

The 132 pin CPGA package can be fitted with a heatsink to allow operating the chip above 20MHz.

Investigation into porting the design into HP-Cmos26 are underway. The standard libraries are well-suited for 50MHz system operation, and the reduced silicon area will translate directly into a lower cost part and larger packaging offerings.

## Conclusions

A large digital signal processing IC has been fabricated in HP-Cmos34. Routing processes have been improved, and the standard cell offering enhanced with additional cells. More accurate four-parameter timing models have been developed for Verilog and other industry simulators. New software was applied in the generation of a large set of test vectors. Sharing the design with the customer was largely successful without major show-stoppers resulting in beta-site quality systems on schedule. Efforts to port the design into HP-Cmos26 are underway promising higher performance and more competitive systems.



Die Size	13.7mm x 14.1mm
Routed Cells	56K gates
Custom RAM	48K bits
Custom ROM	12K bits
Total FETs	540,000
Package	600mil 132-CPGA
Power Supply	5.0V +/- 10%
Operating Power	2.5W (40MHz)

Table 3: Chip Characteristics and Photomicrograph

## An Extended Reed Solomon Decoder Design

J. Chen

NASA Space Engineering Research Center for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

P. Owsley

Advanced Hardware Architectures  
Moscow, Idaho 83843

J. Purviance

NASA Space Engineering Research Center for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract-** It has previously been shown that the Reed Solomon (RS) codes can correct errors beyond the Singleton and Rieger Bounds with arbitrarily small probability of a miscorrect [1]. That is an  $(n,k)$  RS code can correct more than  $(n-k)/2$  errors. An implementation of such an RS decoder is presented in this paper. An existing RS decoder, the AHA4010, is utilized in this work. This decoder is specially useful for errors which are patterned with a long burst plus some random errors.

### 1 Introduction

It is well known that an  $(n,k)$  RS code can correct up to  $(n-k)/2$  random errors. When burst errors are involved, the error correcting ability of the RS code can be increased beyond  $(n-k)/2$  with arbitrarily small probability of a miscorrect [1]. Errors considered in this paper, called composite errors, have a single burst plus random error pattern.

RS codes are powerful error correcting codes. There is a rich history of work developing decoding algorithms for RS codes. Virtually all of the work focuses on the general case of  $t$  unknown error locations. It is possible to extend the error correction capability of a RS code if error location information is available from some external source. This is called erasure decoding.

The extended decoding technique presented in this paper assumes that the locations of the burst are known and treats them as erasures. All possible burst error positions are given to the decoder sequentially as "guesses" to the burst error location. That is, the burst part of the error becomes an erasure and an erasure-locator polynomial is generated from the erasure locations for each burst location guess. By sending this erasure-locator polynomial along with a received code word to a general purpose RS decoder, such as AHA4010, the RS decoder will decode the received codeword. The result outputted by the

RS decoder is either a corrected data or a signal which indicates no correction can be made.

The erasure-locator polynomial is generated iteratively for all possible locations during the decoding procedure. It is possible that more than one error polynomial results from this iterative procedure. When more than one error is obtained, the error that has higher probability of occurrence should be chosen. It is assumed in this paper that an error with smaller weight has higher probability of occurrence. This is true for most channels.

If the chosen error is not the true error, a miscorrection occurs. The probability of miscorrection is a function of the size of the error that is detected and the channel statistics. It is usually very low as shown in reference 1.

The implementation presented in this paper is based on the AHA4010 RS decoder. The purpose is to increase the error correction capability with very little increase on the hardware and software.

## 2 Standard Decoding Description

The standard procedure for decoding the RS code is summarized below:

STEP 1: Compute syndromes

$$S_j = v(\alpha^{j+j_0-1}) \text{ for } j = 1, 2, \dots, 2t.$$

STEP 2: From the syndromes, form the error-location polynomial  $\Lambda(x)$ , where  $\Lambda(x) = (1 - xX_1)(1 - xX_2) \dots (1 - xX_i)$  and  $X_1, X_2, \dots$  and  $X_i$  are the error locations.

STEP 3: Find error location  $X_j$  ( $j = 1, \dots, i$ ) by finding zeros of  $\Lambda(x)$ .

STEP 4: Find error magnitude  $Y_j$  ( $j = 1, \dots, i$ ) by calculating first  $i$  syndrome equations.

STEP 5: Correct the error.

Two polynomials are needed during the decoding and they are:

$$S(x) = \sum_{j=1}^{2t} S_j x^{j-1} \quad (1)$$

and

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}} \quad (2)$$

This second equation is commonly known as the Key Equation, because solving it is the key to decoding the RS code. After obtaining the error locations, the error magnitudes can be found as:



$$Y_t = -\frac{X_r^{j_0-1}\Omega(X_t^{-1})}{\Lambda'(X_t^{-1})} \quad (3)$$

For  $j_0 = 1$ ,

$$Y_t = -\frac{\Omega(X_t^{-1})}{\Lambda'(X_t^{-1})} \quad (4)$$

It is now clear that the decoding procedure becomes one of finding the  $\Lambda$  and  $\Omega$  polynomials from  $S(x)$ , and then finding the location and magnitude of the errors from those two polynomials.

When erasures are involved, an erasure-locator polynomial is created.

$$\Gamma(x) = \prod_p (1 - xX_p)$$

where the  $X_p$ 's are the erasure locations.

The Key equation can be solved for  $\Lambda$  and  $\Omega$  in several ways. One of them is Euclid's recursive algorithm. The Euclid's recursive algorithm is briefly described below. First let

$$\begin{aligned} \Omega^{(-1)}(x) &= x^{2t} \\ \Omega^{(0)}(x) &= S(x)\Gamma(x) \pmod{x^{2t}} \\ \Lambda^{(-1)}(x) &= 0 \\ \Lambda^{(0)}(x) &= \Gamma(x) \end{aligned}$$

the recursive equations are

$$\Omega^i(x) = R_{\Omega^{(i-1)}(x)}[\Omega^{(i-2)}(x)], \quad (5)$$

or equivalently,

$$\Omega^{(i-2)}(x) = q^{(i)}(x)\Omega^{(i-1)}(x) + \Omega^{(i)}(x) \quad (6)$$

and

$$\Lambda^{(i)}(x) = q^{(1)}(x)\Lambda^{(i-1)}(x) + \Lambda^{(i-2)}(x) \quad (7)$$

The recursion is continued until the degree of  $\Omega$  is less than  $t + p/2$ , where  $p$  is the number of erasures.

Erasures are the errors which have been located prior to decoding. Utilizing this information will improve the error correction capability of the decoder. Since the burst is a big part of a composite error, a burst erasure will make the error correction capability much greater. This idea leads to the following approach:

**STEP 1** Set stop conditions, the maximum iteration time  $N$  and  $n=0$ .

**STEP 2** Assume the burst begins at location  $a$  and  $n=n+1$ .

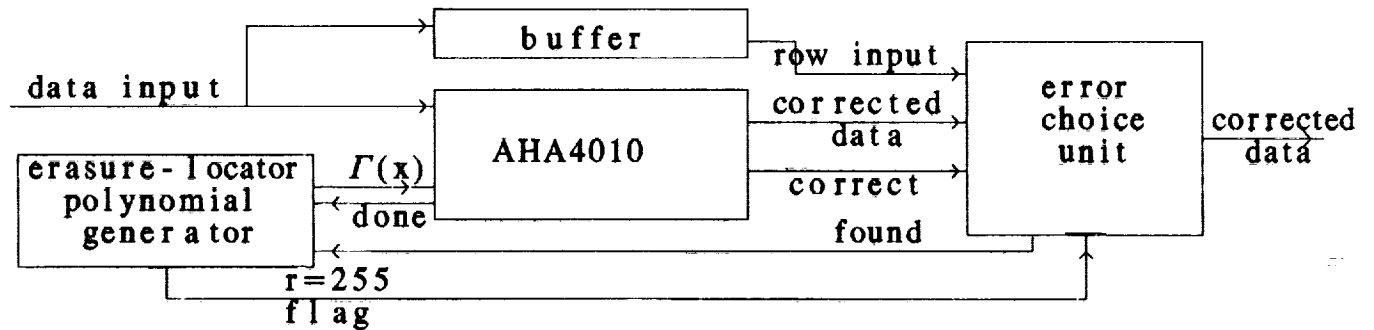


Figure 1: Block Diagram

**STEP 3** Decode the error with the burst as erasures.

**STEP 4** If the result satisfies the stop conditions or  $n_i N$ , go to STEP 5. Else, increase the beginning location of the burst, go to STEP 2.

**STEP 5** Report the result.

In other words, the decoding method, used by the extended decoder, is to guess where the burst part of the error is and try to decode it.

### 3 Extended Decoder Design

The extended RS decoder has an AHA4010 decoder at its center. An erasure-locator polynomial generator, an error choice unit and a data buffer are attached to the AHA4010 decoder. The top level block diagram of this extended decoder is shown in Figure 1.

The erasure-locator polynomial generator generates  $\Gamma(x)$ .  $\Gamma(x)$  could be generated for every possible error location. However, this may not be necessary. For example, let error,  $e(x)$ , be defined as:

$$e_1(x) = \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^0 x^3 + \alpha^4 x^{13} \quad (8)$$

The error,  $e(x)$ , can be interpreted as

$$1. e(x) = 0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^0 x^3 + \alpha^4 x^{13}$$

A burst length of 5 ( $0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^0 x^3$ ) and one random error ( $\alpha^4 x^{13}$ ).

$$2. e(x) = 0x^{-2} + 0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + x^3 + \alpha^4 x^{13}.$$

A burst length of 5 ( $0x^{-2} + 0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2$ ) and two random errors ( $\alpha^0 x^3, \alpha^4 x^{13}$ ).

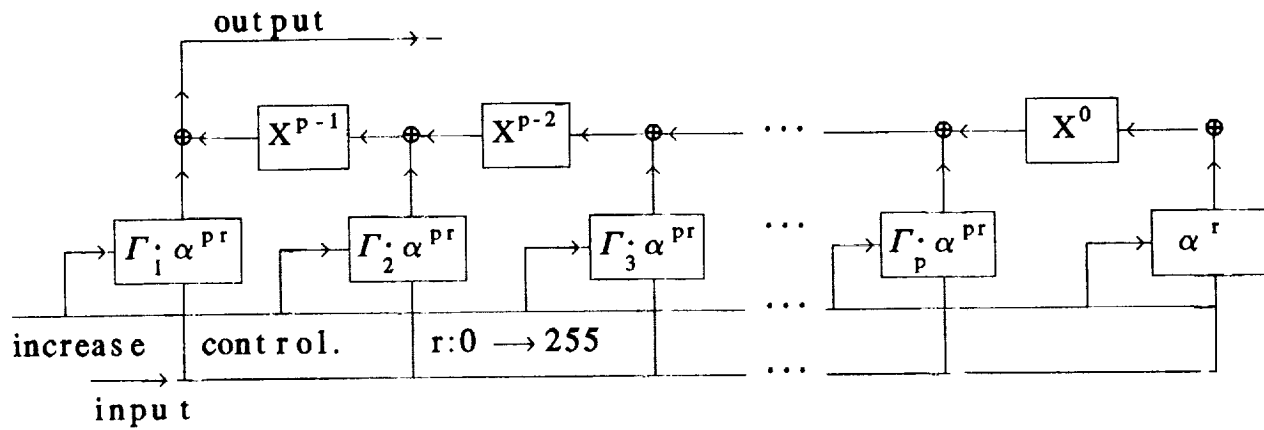
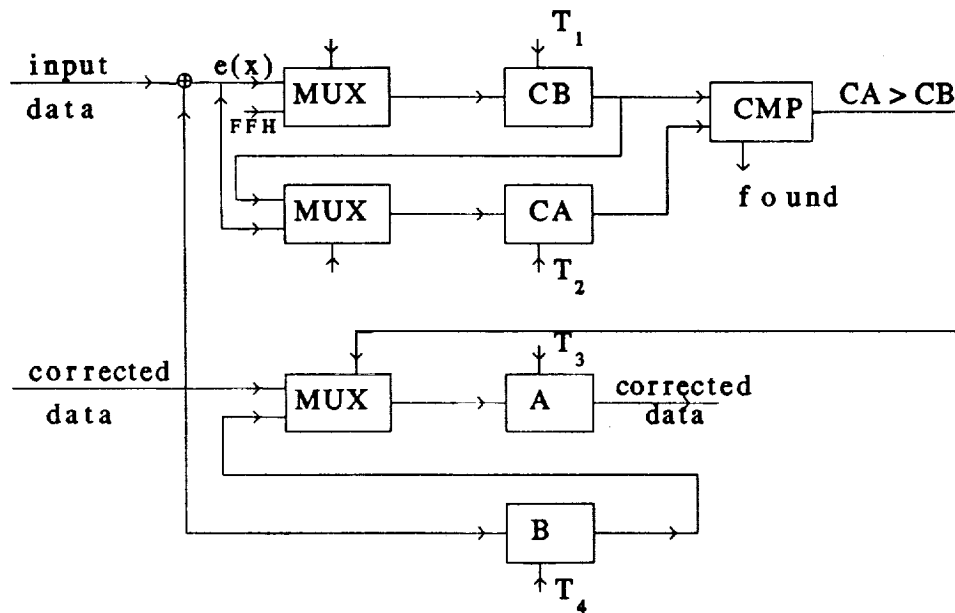


Figure 2: Erasure-locator Polynomial Generator



CONTROLS:

$$T_1 = P_0 + P_1 \cdot \text{CORRECT} \cdot (C \neq 1),$$

$$T_2 = P_1 \cdot \text{CORRECT} \cdot (C = 1) + P_2 \cdot (CA > CB),$$

$$T_3 = P_0 + P_1 \cdot \text{CORRECT} \cdot (C = 1) + P_2 \cdot (CA > CB),$$

$$T_4 = P_0 + P_1 \cdot \text{CORRECT} \cdot (C \neq 1).$$

Figure 3: Error Choice Unit

3.4.6

$$3. e(x) = \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^0 x^3 + 0x^4 + 0x^5 + \alpha^4 x^{13}.$$

A burst length of 5 ( $\alpha^6 + \alpha^9 \alpha^1 + \alpha^6 \alpha^2 + \alpha^0 \alpha^3 + 0\alpha^4$ ) and one random error ( $0x^5, \alpha^4 x^{13}$ ).

$$4. e(x) = 0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^0 x^3 + 0x^4).$$

A burst length of 5 ( $0x^{-1} + \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^0 x^3 + 0x^4$ ) and two random error ( $0x^5, \alpha^4 x^{13}$ ).

A RS code with the ability of correcting a burst of length 5 and 2 random errors will correct all the errors above. Using this logic,  $\Gamma(x)$  can be generated every  $m$  error location bits. The user must decide the value of  $m$  under the consideration of the number of iteration times and the size of the correctable error.

Meanwhile, the error choice unit stores the data corrected by the AHA4010 decoder and reverses it back to the error polynomial. If the size of the error is less than  $t'$  (i.e. This error has the highest probability of occurrence), the error choice unit interrupts the iteration and outputs the corrected data. Otherwise the iteration continues. If more than one error is found, the error choice unit compares these errors and the smallest error is chosen (It is assumed that the smallest error has the highest probability of occurrence).

## 4 Erasure-Locator Polynomial Generator

Assume the received code words have a composite error patterned with  $i$  random errors and one burst error of length  $v$ . The burst locations may be  $\alpha^{r+1}, \alpha^{r+2}, \dots, \alpha^{r+v}$ , where  $r$  is from 0 to 255. The erasure-locator polynomial,  $\Gamma(x)$ , has a form:

$$\begin{aligned} \Gamma(x) &= \prod_{j=1}^v (1 + x\alpha^{j+r}) \\ &= \prod_{j=1}^v (\alpha^{-r} + x\alpha^j)\alpha^r \\ &= \alpha^{vr}(\Gamma_1 x^v + \Gamma_2 x^{v-1} + \dots + \Gamma_v x + \alpha^{-v}) \end{aligned}$$

where  $\Gamma_1, \Gamma_2$ , and  $\Gamma_v$  are constant and  $r$  is from 1 to 255.

For each received code word, the corresponding decoding process is performed  $N/m$  times with  $N/m$  different  $\Gamma(x)$ , where  $N$  is the length of the RS code and  $m$  is the bits that  $\Gamma(x)$  skips. At each end of the decoding process, a DONE signal is sent to the erasure-locator polynomial generator. The DONE signal causes erasures to shift to the right  $m$  bits. Therefore, a new  $\Gamma(x)$  is generated. This operation repeats until a FOUND signal is received or  $r > 255$ .

The erasure-locator polynomial generator is depicted in Figure 2. The coefficients of this polynomial,  $\Gamma_j \alpha^{pr}$  ( $j$  from 0 to  $v$ ), are not constant.  $\Gamma_j \alpha^{pr}$  multiply by  $\alpha$  whenever INCREASE CONTROL (i.e. DONE signal) is assertive.

The operations can be described in a register transfer language where each  $P_i$  is a control state that defines the data transfers that take place when  $P_i$  is active. A register transfer language description for the erasure-locator polynomial generator is shown below:

- $P_0$  :  $r=0$ , if  $GO=1$ , then go to  $P_1$ .
- $P_1$  : if  $FOUND=1$  or  $r=255$ , then go to  $P_0$  , else  $\Gamma_0 = \alpha^r, \Gamma_1 = \Gamma_1 \alpha^{pr}, \Gamma_2 = \Gamma_2 \alpha^{pr}, \dots, \Gamma_p = \Gamma_p \alpha^{pr}$  and  $r = r + 1$ .
- $P_2$  :  $\Gamma(x) = \prod(1 - x \alpha^r)$  , if  $DONE=1$ , go to  $P_1$  .

## 5 Error Choice Unit

During the decoding iteration, it is possible that more than one error results. The error with the highest probability of occurrence should be chosen. It is assumed that will be the smallest error. The diagram of the error choice unit is shown in Figure 3.

The first data corrected by the AHA4010 decoder is stored in register A, its corresponding error is also calculated and the size of the error is stored in CA. If the size of the error is less than  $t'$ , the CMP asserts the FOUND signal and outputs the data in register A. The decoding process otherwise continues. The second corrected data is stored in register B, the size of the second error is stored in CB. The CMP compares the values of CA and CB. If  $CA > CB$ , A is replaced by B and CA is replaced by CB. If the value of CB is less than  $t'$ , the CMP asserts the FOUND signal and outputs the data in register A. If  $CA \leq CB$ , nothing changes. This comparison is performed every time a corrected data is output from the AHA4010 decoder. It guarantees that the register A always has the data which is corrected from the smallest error.

A signal from the erasure-locator polynomial generator tells the error choice unit that the iteration is finished. The data in register A is the output.

A register transfer language description for the error choice unit is:

- $P_0$  :  $0 \rightarrow A, 0 \rightarrow B, 1 \rightarrow C, FFH \rightarrow CB$ , if  $GO = 1$ , go to  $P_1$  .
- $P_1$  : if  $CA < t'$  or  $CB < t'$  or  $FLAG=1$  (i.e.  $r=255$ ), output data, set  $FOUND=1$ , go to  $P_0$  .
- if  $CORRECT=1$  &  $C=1$ ,  $correctedData \rightarrow A$ ,  $size(correctedData) \rightarrow CA$ ,  $c = c + 1$ ;
- if  $CORRECT=1$  &  $C \neq 1$ ,  $correctedData \rightarrow B$ ,  $size(correctedData) \rightarrow CB$ ;
- $P_2$  : if  $CA > CB$ ,  $B \rightarrow A$ ,  $CB \rightarrow CA$ , go to  $P_1$ .

$CORRECT$  is a signal from the AHA4010 decoder which indicates a correction has or has not been made.  $C$  is a counter. It counts the number of correction times for one received code word.

## 6 An Example

Consider a (255,235) RS code over  $GF(2^8)$  defined by the primitive polynomial  $p(x) = x^8 + x^7 + x^2 + x^1 + 1$  with the primitive element  $\alpha = x$ . This code can normally correct ten random errors. Assume received errors have a burst of length 8 and 5 random errors. After considering the number of iteration times and the size of the correctable error, let's set the  $m=4$  and  $t'=11$ .

SOLUTION:

The received polynomial is:

$$v(x) = x^{14} + \alpha^3 x^{15} + \alpha^{200} x^{16} + \alpha^8 x^{17} + \alpha^{40} x^{18} + \alpha^{23} x^{19} + \alpha^6 x^{20} + x^{21} + \alpha^{54} x^{183} + \alpha^{71} x^{198} + \alpha x^{233}. \quad (9)$$

When the extended RS decoder is turned on, the erasure-locator polynomial is:

$$\Gamma(x) = \prod_{j=1}^8 (1 + x\alpha^j). \quad (10)$$

This  $\Gamma(x)$  is sent to the AHA4010 decoder, the FOUND signal is zero. Multiply the coefficients of  $\Gamma(x)$  by  $\alpha^{32}$  (i.e.  $\alpha^{vr} = \alpha^{4 \cdot 8} = \alpha^{32}$ ). The erasure-locator polynomial becomes:

$$\Gamma(x) = \prod_{j=1}^8 (1 + x\alpha^j \alpha^4)$$

and this new  $\Gamma(x)$  is sent to the AHA4010, the FOUND signal is still zero. This decoding process performs repeatedly until the FOUND signal is one. That gives the corrected data:

$$\{0, 0, 0, \dots, 0\}$$

The corresponding erasure-locator polynomial is:

$$T(x) = \prod_{j=1}^8 (1 + x\alpha^j \alpha^{12}) \quad (11)$$

and the corresponding error polynomial is:

$$v(x) = x^{14} + \alpha^3 x^{15} + \alpha^{200} x^{16} + \alpha^8 x^{17} + \alpha^{40} x^{18} + \alpha^{23} x^{19} + \alpha^6 x^{20} + x^{21} + \alpha^{54} x^{183} + \alpha^{71} x^{198} + \alpha^{233}.$$

## 7 Summary

An extended RS decoder has been presented in this paper. With two extra circuits, the error correction capability of a general purpose RS decoder can be increased. This design shows a way to improve the error correction capability of existing RS decoders.

## References

- [1] P. Owsley, "Burst Error Correction Extensions for Reed Solomon Codes," PH.D Dissertation, E.E. Dept. University of Idaho, July 1988.
- [2] W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," IEEE Trans. Inf. Theor. IT-6 (1960), pp. 459-470.
- [3] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, 1983.





# An Improved Distributed Arithmetic Architecture

X. Guo and D. W. Lynn

NASA Space Engineering Research Center for VLSI System Design

University of Idaho

Moscow, Idaho 83843

**Abstract-** Speed requirements have been, and will continue to be a major consideration in the design of hardware to implement digital signal processing functions like digital filters and transforms like the DFT and DCT. The conventional approach is to increase speed by adding hardware and increasing chip area. The real challenge is to save chip area while still maintaining high speed performance. The approach we propose is based on the distributed arithmetic implementation (DA) of digital filters. The improvement is based on two observations. Firstly, a single memory element can replace several identical memory elements in a fully parallel DA implementation. Secondly, truncation or rounding may be introduced into the computation at strategic points without increasing error unduly. Both of these approaches can be used to attain area savings without impairing speed of operation.

## 1 Introduction

Finding the inner product between two vectors is an operation that commonly arises in signal processing as well as in general data processing. Digital convolution and correlation are directly described as inner products. Other operations such as the discrete Fourier transform and other common transforms can be implemented as a sequence of inner products. Consider the inner product

$$y = \sum_{k=1}^K A_k x_k \quad (1)$$

In the case of a FIR digital filter,  $A_k$  represents a set of fixed weights, and  $x_k$  represents the current and past  $K - 1$  filter inputs. The inner product can be implemented directly by using a single multiplier and an accumulator in a serial one product at a time manner, as in Figure 1, or in a fully parallel manner by using  $K$  multipliers and a multi-input adder or adder tree, as in Figure 2. Obviously, the fully parallel architecture will always be faster than the serial approach.

The distributed arithmetic (DA) approach to computing the inner product was developed in the early seventies [1,2,3,4,5,6,7,8]. In this approach, combinations of the  $A_k$  are precomputed and stored in memory. Input data are used to identify which memory words are to be fetched, shifted and added to produce the final result. Without loss of generality,

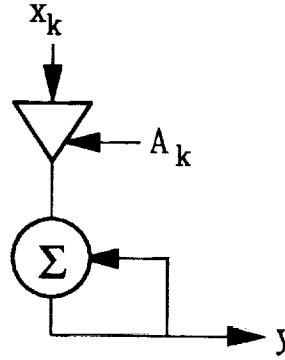


Figure 1: Multiplier-Accumulator Implementation

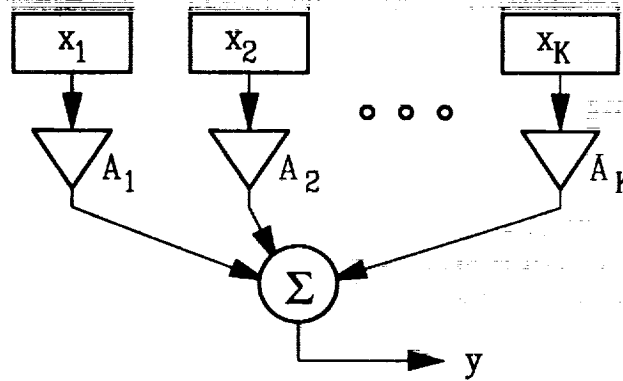


Figure 2: Fully Parallel Direct Implementation

first assume that the  $x_k$  are scaled such that  $|x_k| < 1$ . In two's complement form

$$x_k = -b_{k0} + \sum_{m=1}^{M-1} b_{km} 2^{-m} \quad (2)$$

where the  $b_{km}$  represent the individual bits in  $x_k$  with  $b_{k0}$  the sign bit. Substituting (2) into (1) and rearranging the order of summation gives

$$y = \sum_{m=1}^{M-1} \left\{ \sum_{k=1}^K A_k b_{km} \right\} 2^{-m} - \sum_{k=1}^K A_k b_{k0} \quad (3)$$

Since the bits  $b_{km}$  are either 0 or 1, the term  $\sum_{k=1}^K A_k b_{km}$  can be precomputed for all  $2^K$  possible combinations of  $b_{km}$ . These values are then stored in a ROM or RAM. The actual combinations of  $b_{km}$ , arising out of the input data, are used to address one of the precomputed terms stored in the memory. Note that these combinations are formed by selecting the  $m$ th bit from each of the  $K$   $M$ -bit input words. The  $m$ th term so addressed is then shifted by  $m$  bits to the right before being added to the other  $M$  terms. The only exception to this is when  $m = 0$ . In this case, which corresponds to using the sign bits of the input data to form the address, the addressed term is subtracted from the other terms.

As with the direct implementation of the inner product, there are two approaches to implementing DA. The inner product can be computed by using only one memory and a single accumulator as shown in Figure 3, or in a fully parallel manner by using  $M$  memories,

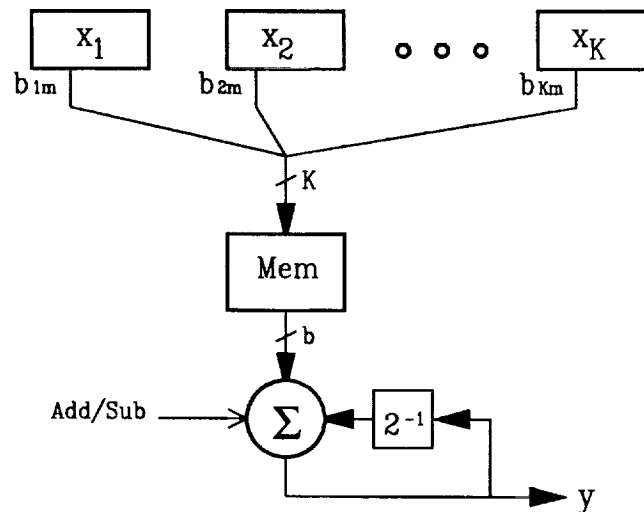


Figure 3: Single Memory DA Implementation

each with identical contents, as shown in Figure 4. Again, the fully parallel approach will always be the fastest. In Figure 4, we note that the shifting is actually accomplished by connecting the memory outputs to appropriate positions on a multi-input adder.

Comparing Figures 4 and 2 is instructive. We note that where the input data words are  $M$  bits wide,  $M$  memories are always used in the DA implementation, independent of  $K$ , the number of multiplies in the inner product. However, each memory must store  $2^K$  terms. So, increasing  $K$  will increase the required size of the memories. Also, as  $K$  increases, the number of stored bits per term must increase in order to maintain accuracy. The direct implementation, by comparison, uses  $K$  multipliers. As  $M$  increases, the width and depth<sup>1</sup> of the multipliers must increase to preserve accuracy. Thus, depending on the word size, number of products, and required accuracy, one approach may have size advantages over the other.

In terms of speed, DA does have one clear advantage over the direct implementation. Increasing the accuracy of the inner product by increasing the number of bits in the input data words *and* in the coefficients will not degrade the speed performance of a DA implementation. The number of memories and the width of each will increase, but the number of stored terms in each memory will not. In a direct implementation, however, not only the width of the multipliers increase, but so will their depth resulting in slower performance. Increasing  $K$  does not decrease the speed of the multipliers, but it will increase the depth of the adder tree in the direct implementation resulting in some loss of performance. In a DA implementation, increasing  $K$  will slow down the memories, but it does not increase the depth of the adder tree.

While the structure of the fully parallel DA implementation is very regular and hence attractive for VLSI implementation, it appears to be very inefficient in terms of its use of space. That is, for each inner product computed, only one of the  $2^K$  terms stored in each memory is used. Further, the contents of each of the  $M$  memories is *identical*. Our

<sup>1</sup>This assumes that the width of the coefficients also increases proportional to  $M$ .

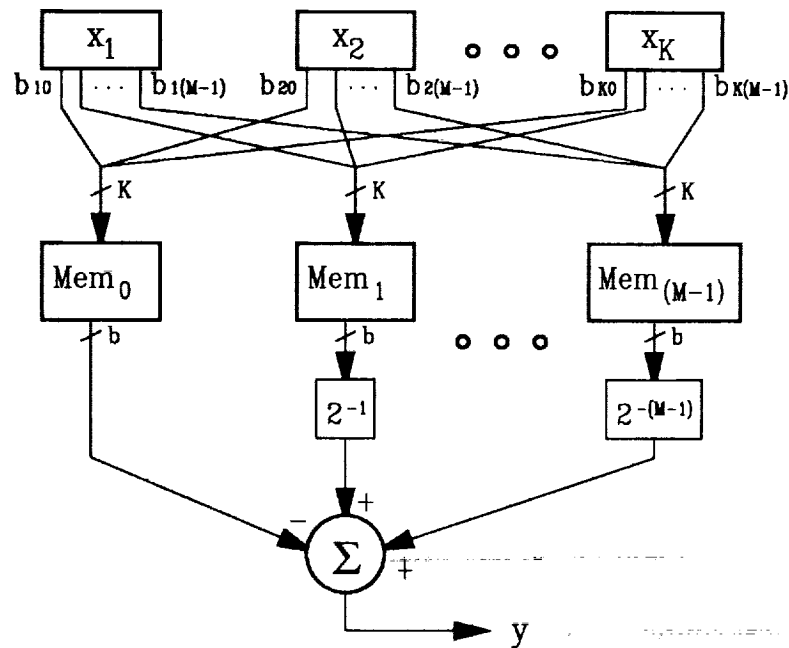


Figure 4: Fully Parallel DA Implementation

first observation about how the fully parallel DA architecture may be improved involves replacing the  $M$  memories with just one memory unit that provides  $M$  data access paths.

## 2 An Improved DA Architecture

A ROM, using one transistor per storage bit, is shown in Figure 5. The stored bits are zero or one depending on whether the drain of the associated transistor is connected to the data line or not. Note that the address decoder and the data line sense amplifiers are not shown. Not counting these components, the number of transistors required for the memories in a ROM based fully parallel DA implementation is

$$n_t = M(b2^K + 2^K + b) \quad (4)$$

where  $b$  represents the number of bits stored in each word of the memory. Next, consider Figure 6 which represents one plane of a  $M$ -way multi-access memory. Each plane stores one word and  $2^K$  planes together make up the complete memory unit as shown in Figure 7. Each plane has  $M$  sets of  $b$  control transistors that are used to route the stored word to the appropriate output register. Each set of  $b$  control transistors is controlled by a single control line. The data bits associated with control line  $m$  in each plane are connected to a bus which connects with output register  $m$ . Which of the  $2^K$  control lines is asserted is determined by address decoder  $m$ . Since this circuit effectively addresses the output registers instead of the stored words, there is no need for address lines for the stored words themselves. Further, a transistor is not required for each stored bit. A zero is stored simply with a shorted line, a one with an open. The control transistors assume the function of the

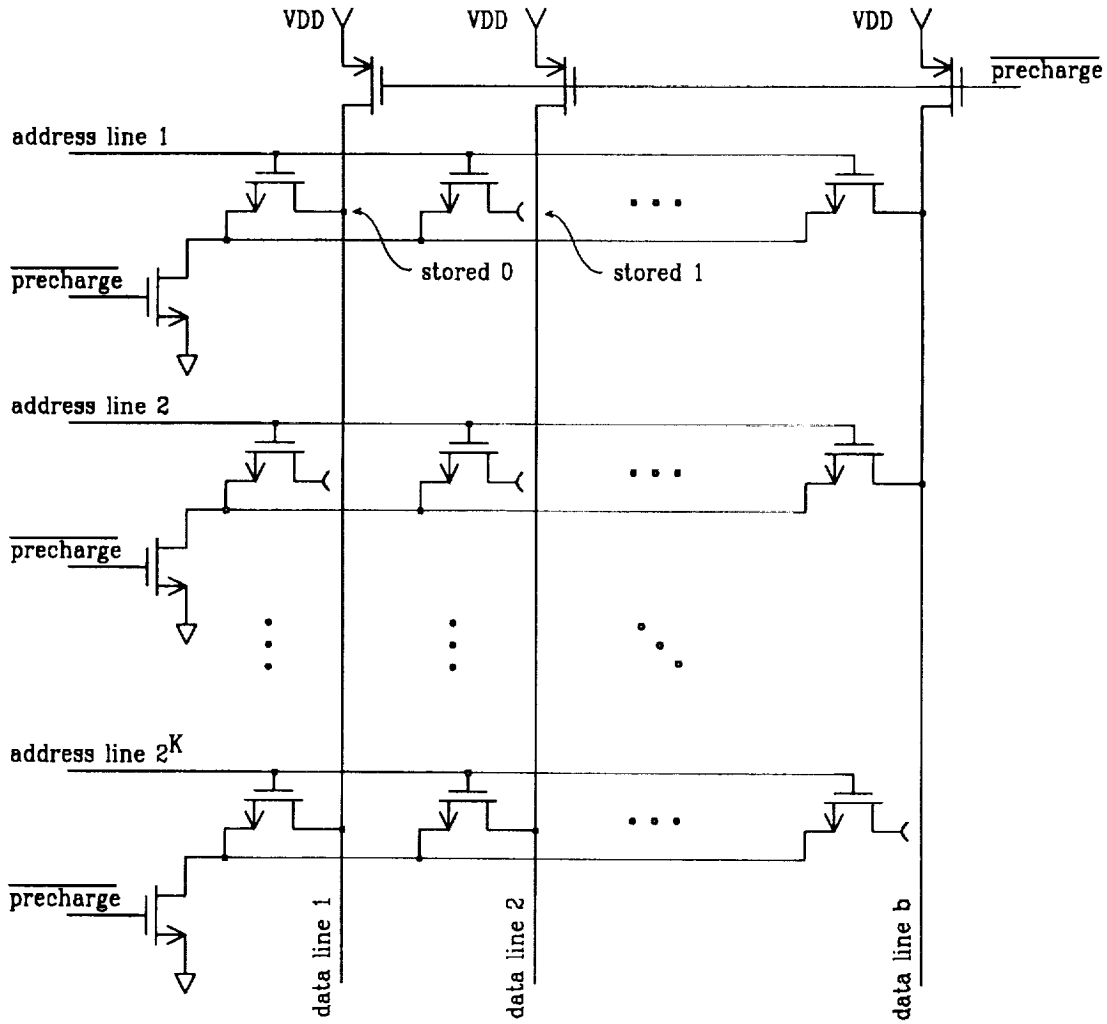


Figure 5: ROM Architecture

storage transistors in the ROM architecture, and provide a path between every stored word in the memory unit and every output register. The total number of transistors required to implement the memory unit, again excluding address decoders and output registers is

$$n_t = Mb2^K + b2^K + 2^K \quad (5)$$

Note that both approaches use  $M$   $K$  to  $2^K$  decoders and identically sized adder trees. The ratio of the number of transistors in the storage sections of the Multi-Access memory unit and the memories in a fully parallel DA architecture give an estimate of the area savings potential presented by one approach over the other. Dividing (5) by (4) gives

$$R_{area} = \frac{Mb2^K + b2^K + 2^K}{Mb2^K + M2^K + Mb} = \frac{Mb2^K + (b+1)2^K}{Mb(2^k + 1) + M2^K} \quad (6)$$

Since  $b$  will usually be greater than  $M$ ,  $R_{area} > 1$ . That is, the multi-access memory architecture presents no area savings, despite the fact it replaces  $M$  copies of each stored

### 3.5.6

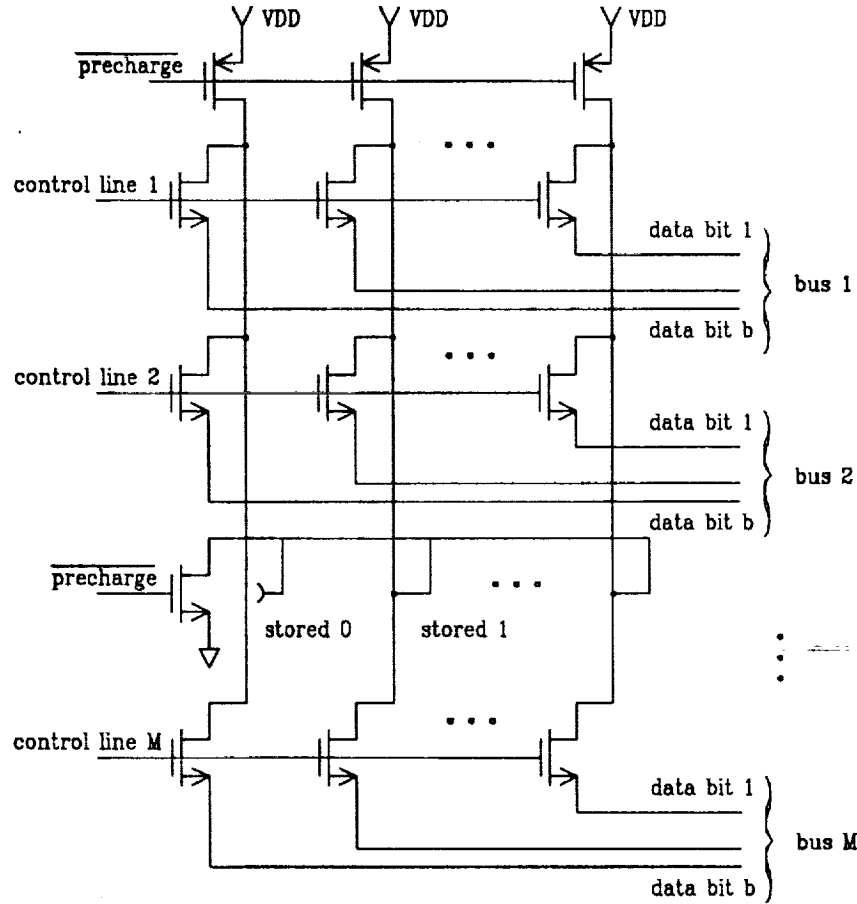


Figure 6: Memory Unit Plane

word with just one copy. This is because the expense of bus controls erases the savings of memory transistors. In fact the number of transistors associated with stored bits in the fully parallel implementation is  $Mb2^K$ . This is also the number of bus control transistors in the multi-access memory. However, for memory architectures where there is more than one transistor per stored bit, the savings in storage transistors will not be absorbed by bus control transistors. To see this, consider the case that 4 transistor static RAM cells are used as memory elements. Static RAM may be required in cases where the inner product is to be configurable, in the sense that the coefficients may be changed from time to time, requiring the memory contents to be rewritten. The conventional static RAM architecture is shown in Figure 8 and one plane of the multi-access memory architecture is shown in Figure 9.

A fully parallel implementation of DA using  $M$  static RAMs of the type shown in Figure 8 would use  $4Mb2^K$  transistors and  $Mb2^K$  cell select transistors. The static RAM multi-access memory unit would use  $4b2^K$  transistors for storage and  $Mb2^K$  bus control transistors. Thus, when static RAM cells are used,

$$R_{area} = \frac{(M + 4)b2^K}{5Mb2^K} = \frac{M + 4}{5M} \quad (7)$$

Here there will be an area savings so long as  $M$  (the number of bits in the input data words and the number of memory units in the fully parallel DA implementation) is greater than 1.

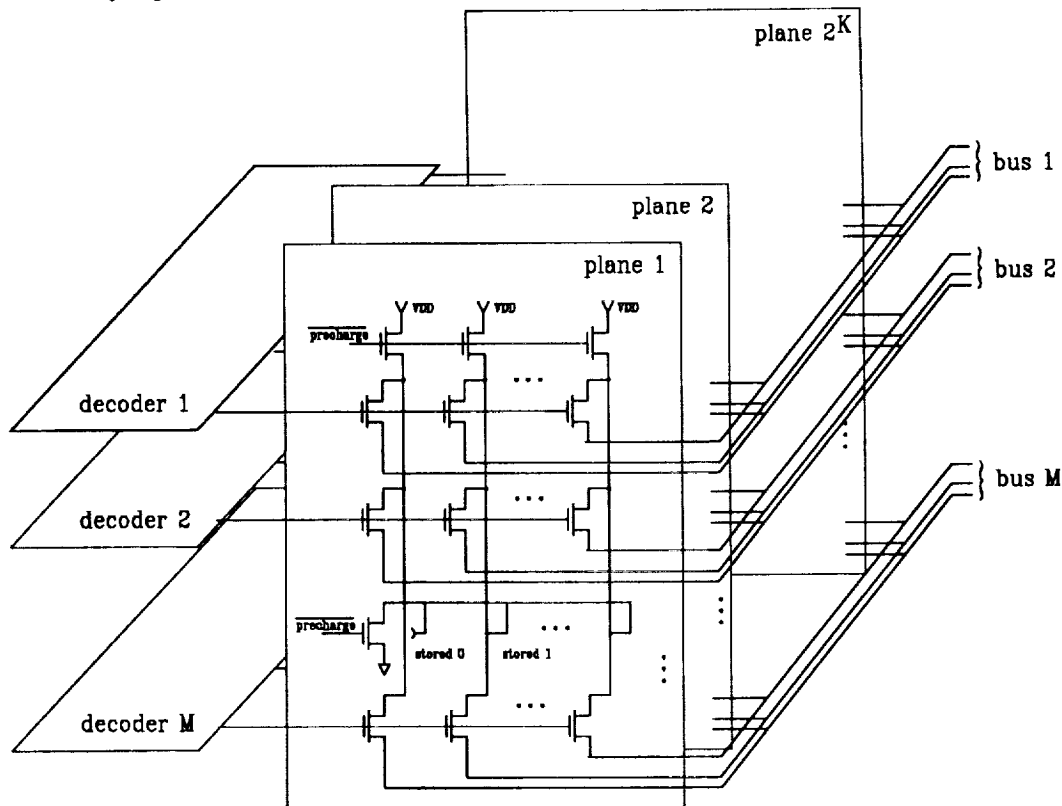


Figure 7: Multi-Access Memory Unit

If  $M = 8$ ,  $R_{area} = .30$ . Thus the area savings can be significant. These observations have been made in the context that the number of transistors corresponds to area requirements. The same results hold whenever the area required for the storage cell (be it transistor based or not) requires more area to implement than do bus control transistors. Again, it must be noted that while the area required for address decoders and the adder tree are the same for both implementations, these requirements are not included in the  $R_{area}$  computation. So,  $R_{area}$  only reflects the savings potential in the storage section of the implementation. To the degree that the storage section dominates the other elements of the implementation this may translate into significant savings. Not only do the other elements need to be included in the computations, but an actual VLSI layout of a multi-access memory based DA circuit needs to be attempted to make sure that the connection complexity of the multi-access memory unit does not overwhelm what appears to be a significant area savings potential in the case of static RAM memory cells.

### 3 Truncation and Rounding

Once each term is fetched from the memory, they are shifted and added to form the final result. This operation is diagrammed in Figure 10. If each memory in a fully parallel implementation stores terms that are  $b$  bits wide, then the resulting inner product will occupy

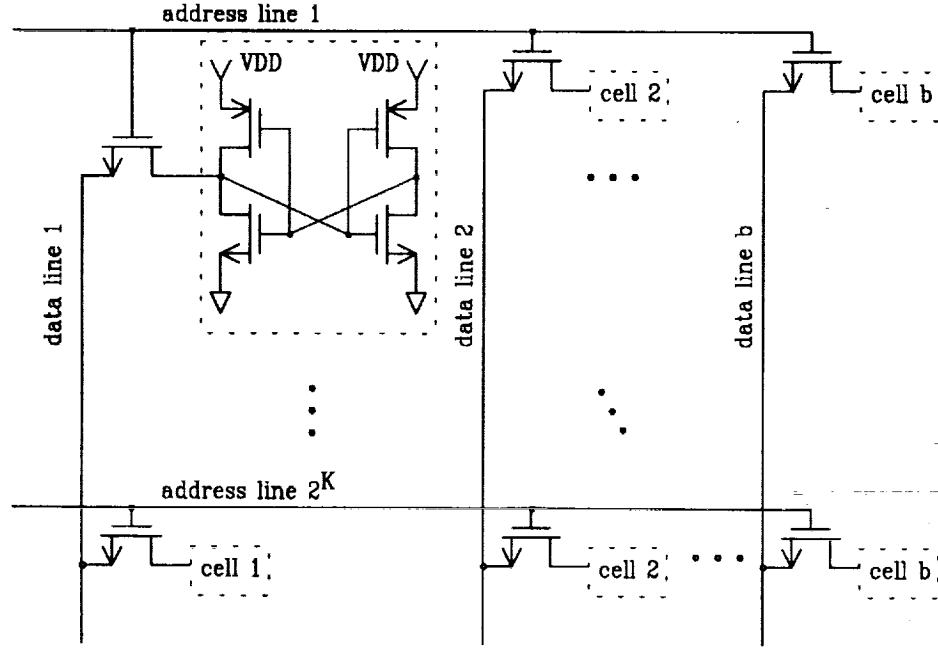


Figure 8: Static RAM

at most  $M + b$  bits.<sup>2</sup> Suppose, however, that the product only needs to be determined to an accuracy of  $f$  significant bits. In this case it may be possible to truncate or round the individual terms before adding. Doing so would not only reduce the amount of hardware required in the adder tree, but would also reduce the size of some of the memories. This is obvious in the case of the fully parallel DA implementation and, as we will see later, it is also true for the multi-access memory unit based implementation. First let us consider what the impact of truncating or rounding will be on the accuracy of the final result.

Truncating the individual terms and discarding bits that fall in column  $f + e$  and to the right (as shown in Figure 10) will give a maximum worst case error of

$$E_{tt} = 2^{-e}((M + b - (f + e)) - 1 + 2^{-(M+b-(f+e))}) \quad (8)$$

where we have normalized the result so that the binary point falls just to the left of column  $f$ .<sup>3</sup> The worst case truncation error is calculated by considering that all the truncated bits are ones.

When we round the individual terms and then discard bits that fall in column  $f + e$  there are two worst case error situations. If the bits in column  $f + e$  are all ones, and all bits to the right are zeros. In this case the error is

$$E_{rta} = 2^{-(e+1)}(M + b - (f + e)) \quad (9)$$

<sup>2</sup>This can be shown by temporarily treating the terms as whole integers and assuming that all  $M$  terms take on the maximum value  $(2^b - 1)$ . The final sum will then be  $(2^b - 1) * (2^M - 1)$  which can be written as  $((2^{b+M} - 1) - (2^M - 1) - 2^b) + 1$ . When written this way and assuming  $M < b$  it is easy to see that the result occupies at most  $M + b$  bits.

<sup>3</sup>We also need  $f \geq b$ .



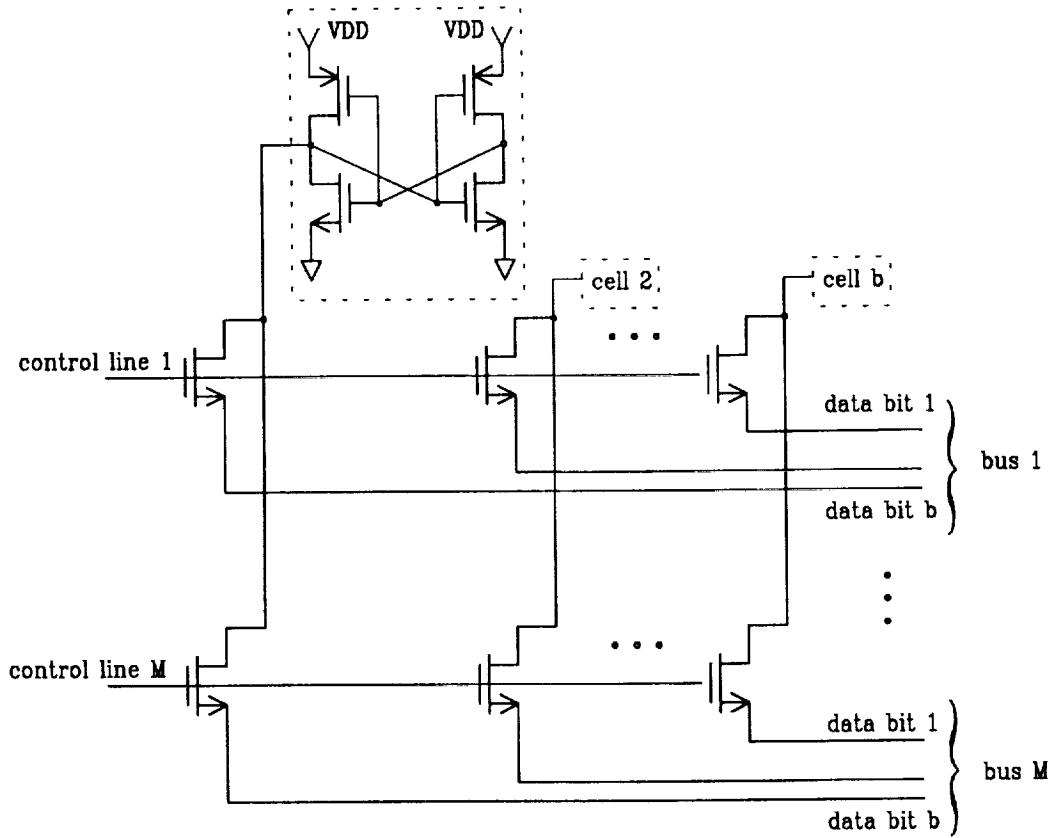


Figure 9: Static RAM Multi-Access Memory Plane

If the bits in column  $f + e$  are zeros and all the bits to the right are ones, the error is

$$E_{rtb} = 2^{-(e+1)}((M + b - (f + e + 1)) - 1 + 2^{-(M+b-(f+e+1))}) \quad (10)$$

After the individual terms have been truncated or rounded to  $f + e$  bits, the final sum is computed and then either truncated or rounded to  $f$  bits. This second truncation or rounding will add to the total error. In the case of truncation, the worst case error will be

$$E_{tf} = 1 - 2^{-(e-1)} \quad (11)$$

In the case of rounding, the additional worst case errors are

$$\begin{aligned} E_{rfa} &= 2^{-1} \\ E_{rfb} &= 2^{-1} - 2^{-(e-1)} \end{aligned} \quad (12)$$

Noting that  $E_{rta} > E_{rtb}$  and  $E_{rfa} > E_{rfb}$ , we will use  $E_{rta}$  and  $E_{rfa}$  when referring to rounding. There are four possible approaches to arriving at the final result depending on which of truncation or rounding is applied to the individual terms and which is applied to the final sum. The four possibilities are summarized in Figure 11. From the graph, we see that as few as five or six extra bits beyond  $f$  are required in order to arrive at errors that are very near what we would expect if we retained all the bits in the individual terms,

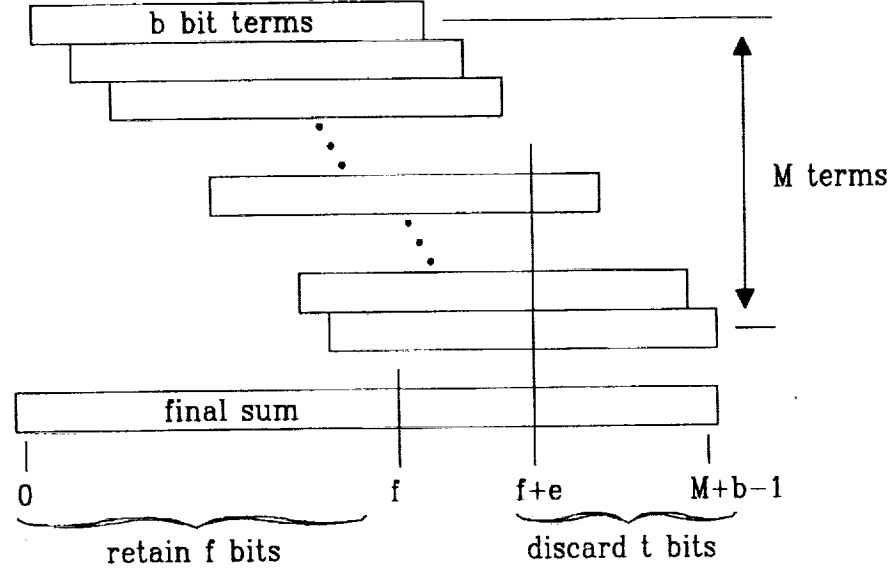


Figure 10: The Final Sum

formed the sum and rounded or truncated the result to  $f$  bits. We also note that this is true independent of whether the individual terms are first truncated or rounded. When  $e < 4$ , the error resulting from rounding is similar to the error resulting from truncating one less bit. These observations suggest that there is not a great deal to be gained by rounding individual terms over truncating them. In the case of a fully parallel DA implementation, the rounding of individual terms can be precomputed and only the rounded terms stored, so there is no cost in doing so. However in the multi-access memory based implementation, rounding of individual terms would have to be performed upon access. As we shall see shortly, the area requirement of the multi-access memory for rounding will be the same for a memory the truncates one less bit. This, coupled with the above observations on error, indicate that rounding individual terms does not provide a very great advantage over truncation in the multi-access memory.

## 4 Implementing Truncation and Rounding

First we consider the transistor cost of a ROM based fully parallel DA implementation. From Figure 10 we see that if we desire to compute the final sum to  $f + e$  bits, we will need  $(M - t)$  ROMs storing  $b$  bit words, and  $t$  ROMs that each store one bit less in succession where  $t = (M + b) - (f + e)$ . Note that for consistency,  $t < b$  and  $t \leq M$ . If  $t \geq b$ ,  $M$  should be reduced and if  $t > M$ ,  $b$  should be reduced. Now, referring to Figure 5 we see that for each bit truncated,  $2^K + 1$  transistors are saved. Since  $\sum_{i=1}^t i = t(t + 1)/2$ , the cost of the implementation is

$$n_t = M(b2^K + 2^K + b) - \frac{t(t + 1)}{2}(2^K + 1) \quad (13)$$

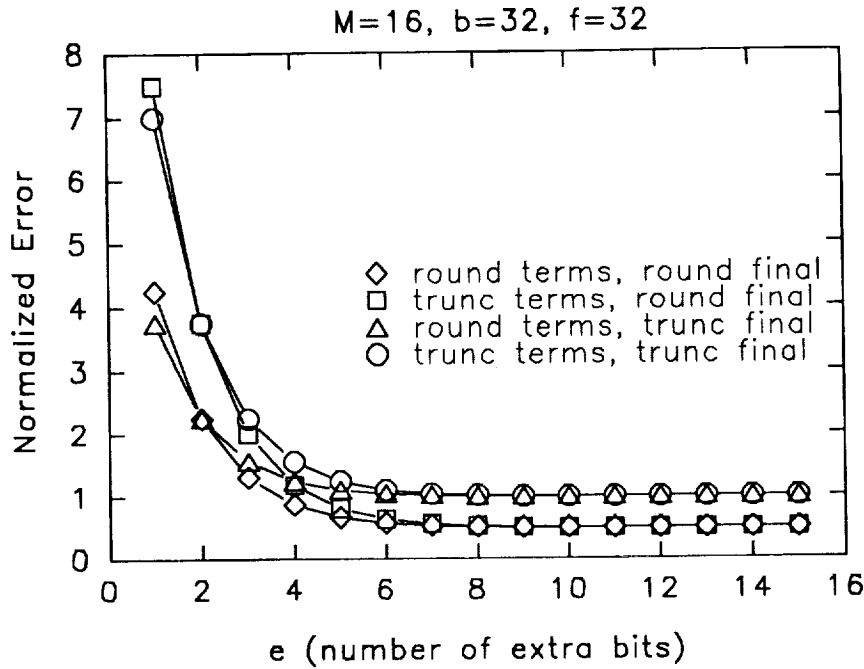


Figure 11: Errors from Truncation and Rounding

Again,  $K$  to  $2^K$  decoders are required for all  $M$  ROMs. We also note that this equation applies equally well to both truncation and rounding of individual terms. Next we consider the transistor cost of the multi-access ROM based implementation. We note that while each stored term must have the full  $b$  bits, the width of the last  $t$  data paths decreases by one bit for each path. This is shown in Figure 12 where the storage cells are implemented by connections (or the lack thereof) to ground through a precharge transistor as in Figure 7. Thus, we save  $t(t+1)/2$  bus control transistors in each plane so, the overall cost of the implementation is

$$n_t = Mb2^K + b2^K + 2^K - \frac{t(t+1)}{2}2^K \quad (14)$$

Comparing the savings of the two approaches, we see, not surprisingly, that the multi-access ROM continues to lose ground against the fully parallel implementation. The disadvantage is further amplified when we consider applying rounding to individual terms. In the fully parallel approach, the rounding is precomputed, but in the multi-access approach the rounding must be computed on-line. An extra bit in each of the terms to be rounded is required. If the bit is a one, the one is to be added to the next more significant bit. This could be achieved by routing the extra bit to an appropriate place in the adder tree. Another approach might be to truncate so that a final sum of  $f + e + 1$  bits is computed, resulting in an equivalent error. In either case, an extra bit would be needed for each of the  $M$  data paths in each of the  $2^K$  planes.

Extending the comparison to the use of static RAM, from Figures 8 and 10, we see that truncating or rounding so that the final sum is computed to  $f + e$  bits would save  $(5t(t +$

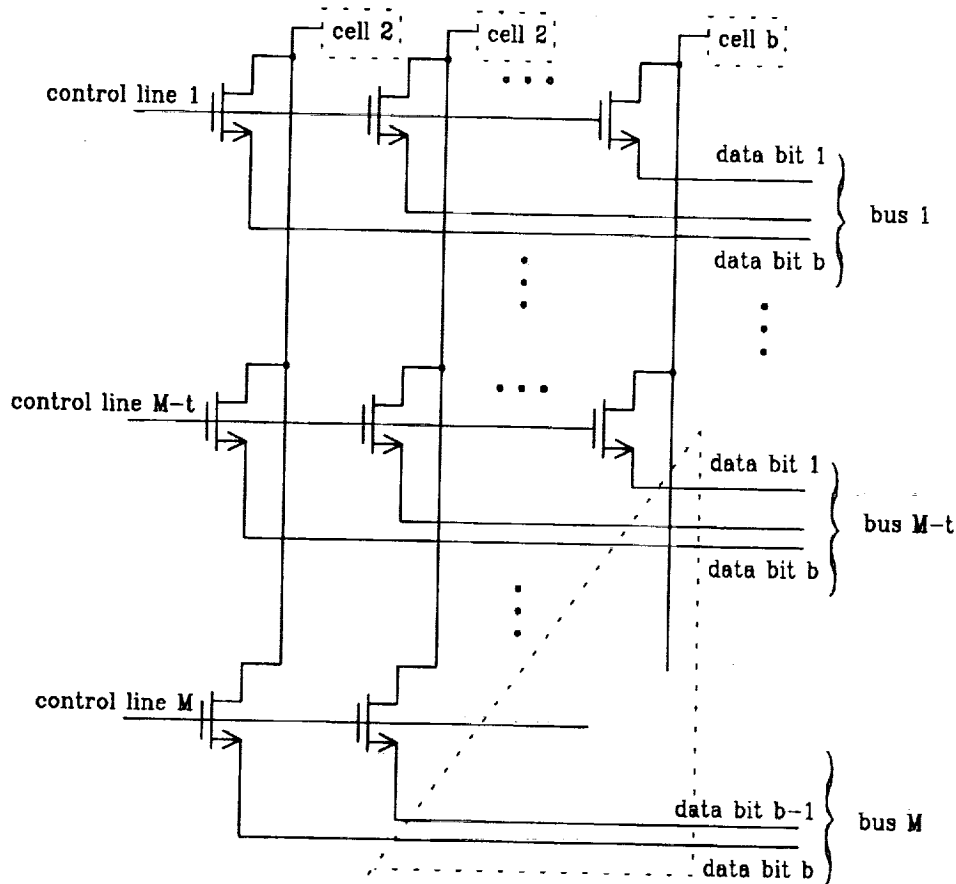


Figure 12: Multi-access Plane with Truncation

$1/2)2^K$  transistors in the fully parallel implementation. Letting the storage elements in Figure 12 be the 4 transistor cells used in Figure 9, we see that the savings from truncation in the multi-access RAM based implementation is the same as it was in the multi-access ROM, namely  $2^K t(t+1)/2$ . Not surprisingly, we find that the fully parallel RAM based implementation benefits more from truncation than does the multi-access based RAM architecture. We note however, that it still possesses a significant advantage. The ratio of the number of transistors becomes

$$R_{area} = \frac{(M+4)b - t(t+1)/2}{5(Mb - t(t+1)/2)} \quad (15)$$

With  $t = M = 8$  and  $b = 16$ ,  $R_{area} = 0.34$  as compared to the .30 ratio that arises if  $t = 0$ . We also note that the reduced number of bus control transistors and reduced bus widths reduces the connection complexity of the multi-access architecture.

## 5 Conclusions

We have shown that the multi-access architecture requires significantly less area than a fully parallel architecture when the number of transistors per stored bit is greater than one, as it will be when static RAM cells are employed. Since this observation is based on the assumption that the transistors used for storage are the same size as those used

for bus control, we can say more abstractly that the multi-access architecture will save space whenever the area required to implement each storage cell is greater than the area required to implement a bus control or routing transistor. The savings estimates do not include the cost of decoders and the cost of the adder tree (which will be the same in both cases). The area requirements of these elements must be added in so we can truly assess the area savings advantage of our approach. Both approaches appear to be fairly regular, both lending themselves well to VLSI implementation. Again this observation is made independently of the implementation of the decoders and the adder tree. The connection complexity between these elements in both architectures also needs to be considered. In short, a VLSI layout of both architectures needs to be done in order to be able to accurately compare the two.

We have also presented the errors associated with truncating or rounding individual terms and the area savings that can result in both architectures from doing so. These errors need to be reconsidered, placing them in the overall context of the inner product. In particular we have not considered what  $b$  should be given  $M$  and  $K$ .

## References

- [1] A. Peled and B. Liu, "A New Approach to the Realization of Nonrecursive Digital Filters," *IEEE Trans. Audio and Electroacoustics*, Vol. AU-21, No.6, pp. 477-485, December 1973.
- [2] S. Zohar, "New Hardware Realization of Non-recursive Digital Filters," *IEEE Trans. on Computers*, Vol. C-22, pp. 328-338, April 1973.
- [3] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-22, pp. 456-462, December, 1974.
- [4] C.S. Burrus, "Digital Filter Structures Described by Distributed Arithmetic," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No.12, pp. 674-680, December, 1977.
- [5] F.J. Taylor, "An Analysis of the Distributed Arithmetic Digital Filter," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-34, No.5, pp. 1165-1170, October 1986.
- [6] M. Hatamian, et al., "Parallel Bit-Level Pipelined VLSI Designs for High-Speed Signal Processing," *Proceedings of IEEE*, Vol.75, No.9, pp.1192-1202, September 1987.
- [7] S. Zohar, "A VLSI Implementation of a Correlator/Digital Filter Based on Distributed Arithmetic," *IEEE Trans. on A.S.S.P.*, Vol. ASSP-37, No.1, pp. 156-160, January 1989.
- [8] S.A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Magazine*, Vol.6, No.3, pp. 4-19, July 1989.
- [9] Xiaoyi Guo, "An Improvement on Distributed Arithmetic Implementation of High Speed Digital Filters", Masters Thesis, University of Idaho, May 1991.

THESE DOCUMENTS SONT LA PROPRIETE DE LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

LES DOCUMENTS SONT PRETES A ETRE CONSULTES PAR LE PUBLIC A LA BIBLIOTHEQUE DE LA MAIRIE DE MONTREAL

# An Analog Retina Model for Detecting Dim Moving Objects Against a Bright Moving Background

R. M. Searfus, M. E. Colvin, F. H. Eckman,  
J. L. Teeters, and T. S. Axelrod  
Lawrence Livermore National Laboratory  
Livermore, California 94550

**Abstract** – We are interested in applications that require the ability to track a dim target against a bright, moving background. Since the target signal will be less than or comparable to the variations in the background signal intensity, sophisticated techniques must be employed to detect the target. We present an analog retina model that adapts to the motion of the background in order to enhance targets that have a velocity difference with respect to the background. Computer simulation results and our preliminary concept of an analog “Z” focal plane implementation are also presented.

## 1 Introduction

We are interested in air and spaceborne surveillance applications that require real-time target detection and tracking against a moving earth background. The scene observed from the surveillance platform may range from a dark earth to bright sunlit clouds and terrain, and the variation in the intensity of a single scene may span several orders of magnitude. As long as the target intensity is sufficiently larger than the variations in the background intensity, simple image processing techniques such as spatial filtering and thresholding can produce satisfactory results; however, for the case of a dim target against a bright, moving background, simple processing methods may produce unacceptable levels of false detections or may completely fail to detect the target.

One approach for reliably detecting and tracking targets under these conditions is to subtract the moving background from the scene, leaving only those objects that have a different velocity than the background. Since the background motion may not be known *a priori* and may change throughout the course of observation, it is important for the sensor to have the capability of adapting to changes in the background velocity. The number of detector signals that must be simultaneously processed<sup>1</sup> imposes a computational demand that exceeds the capability of conventional computer hardware. Furthermore, for a space environment, low-power consumption and compact size are extremely important design constraints.

In this paper, we present a model for an analog retina that adapts to the motion of the background and enhances objects having a velocity difference with respect to the background. A computer simulation of this model is described, and our experience of using the simulation on real and synthetic data is discussed. We also describe a real-time implementation of our model on a PIPE image processing computer, and present a

---

<sup>1</sup>A minimum detector array of 128x128 pixels is required; an array of 512x512 pixels is desired.

mapping of our model to a "Z" focal plane (Z-plane) technology [?] implementation that addresses the real-time processing requirements and the design constraints for space-based operations.

## 2 An Analog Retina-like Model

Very sensitive, high-resolution electronic imaging systems exist with capabilities that surpass those of any biological system. However, current electronic imaging systems do not possess the robustness of a biological system when confronted with a diverse environment, and also lack the real-time processing power of even the simplest vertebrate retina. For the relatively simple task of identifying and tracking moving objects, man-made devices fall short of the biological systems they are designed to mimic.

The goal of our research effort has been to extract and understand the engineering principles underlying natural vision systems and to apply that knowledge to designing better image processing hardware. We are focusing on the retina because research has shown that some animals possess enough image processing "wetware" to detect and track moving objects using only a thin layer of cells at the back of the eyecup (the retina).

The vertebrate retina is more than just a simple light sensor. It is a complex sensor-processor device that transforms the incoming light signal before transmitting it to the visual cortex and other subcortical regions. The retina's full range of functions are presently unknown, but it is clearly involved in dynamic range adjustments, edge enhancement, color preprocessing, and change detection. The retina has five main cell types (photoreceptors, horizontal cells, bipolar cells, amacrine cells, and ganglion cells) and two synaptic layers, the inner and outer plexiform layers, where the processes of these retinal cells interact to produce nontrivial signal transformations. The outer plexiform layer handles spatial processing and dynamic range adjustments, while the inner plexiform layer is involved in change detection and temporal processing. A detailed description of the anatomy and physiology of the vertebrate retina can be found in Dowling [?]. We must emphasize that we are not trying to duplicate the biological retina. Rather we have borrowed several design principles from the retina (especially the outer plexiform layer) to solve a specific image processing problem.

Our model consists of three major components as shown in the block diagram of Figure 1: an artificial retina, augmented by a background removal network, and an image enhancement network. Processing throughout the model is performed on analog data, eliminating the need of analog-to-digital and digital-to-analog conversion.

The artificial retina is based on our previous work involving the use of a retina-like model for detecting moving objects against a fixed background [?], and consists of two parts: a photodetector array analogous to photoreceptors found in biological vision systems; and an image conditioning network that mimics the function of horizontal and bipolar cells in the biological retina. The photodetector array is a mosaic of photosensitive devices that convert light into an electrical signal. Unlike a CCD which produces discrete frames of time-averaged data, the photodetector array produces a continuous, time-varying image.



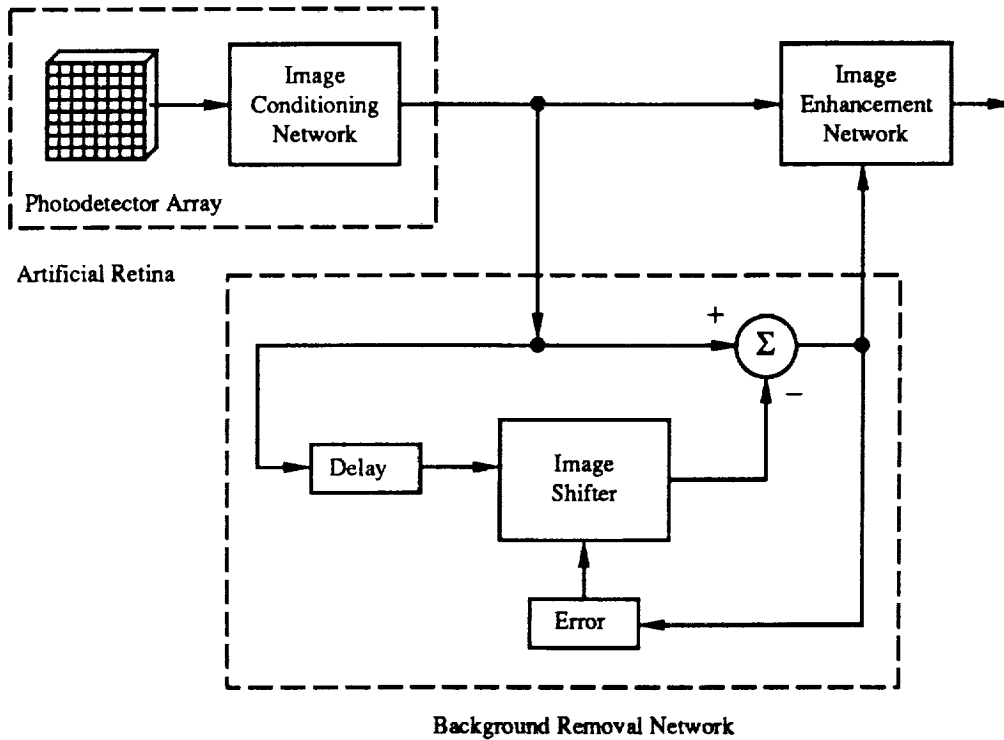


Figure 1: This block diagram shows the three major components of our model: an artificial retina; a background removal network; and an image enhancement network. The artificial retina converts light into a continuous, time-varying image, and conditions this image with amplification and spatial-temporal noise reduction. The background is removed by network layers which subtract a shifted, time-delayed image from the conditioned image, and the result is used to enhance the output of the artificial retina. Further analog and digital processing can be performed on the enhanced image to meet application-specific requirements.

#### 4.1.4

The output of the photodetector array is amplified by the image conditioning network, which also provides temporal and spatial noise reduction.

The background is subtracted from the artificial retina output by the background removal network. To perform this operation, output from the artificial retina is first delayed by a low-pass temporal filter (depending on the application, this delay can either be fixed or variable). The delayed image is then spatially shifted by a neural network layer. The adjustable weights of this image-shifting network determine the total spatial displacement. A difference image is then formed by subtracting the delayed, shifted image from the artificial retina output. If the weights of the image-shifting network are adjusted correctly, the background in the image shifter output will be aligned with the background in the artificial retina output, and the backgrounds will cancel in the difference. Objects having a velocity difference with respect to the background will leave a negative trace at the trailing end of the object and a positive trace at the leading end.

An error feedback network modifies the weights of the image-shifting network to achieve and maintain background alignment. The drift error is determined by rectifying the difference image and summing the rectified values (1).

$$|E|^2 = \int_{Image} (I(\vec{r}, t) - I(\vec{r} + \vec{offset}, t + delay))^2 \quad (1)$$

As the backgrounds are shifted towards alignment, the error decreases; as the backgrounds are shifted away from alignment, the error increases. In a one-dimensional case, the derivative of the error with respect to the spatial shift will determine the required shift direction necessary to bring the backgrounds into alignment (see Figure 2). For a two-dimensional case, the gradient of the error with respect to the X/Y spatial shift will determine the shift direction. The magnitude of the gradient scaled by a feedback gain can be used to determine the shift distance. To allow a more detailed analysis of the error feedback, we have been studying images moving in a single dimension only.

It is possible to estimate the limits on the accuracy of the offset determination due to the use of this simple, aggregate error signal. (More complex and computationally expensive shift error measures are possible, such as calculating a pixel-by-pixel brightness correlation function). To determine the effect of the background clutter, we can compute the change in the sensitivity of the error signal for different background spatial frequencies. If we assume the background is a one dimensional sinusoidal grating, then the magnitude of the error signal (as a fraction of its maximum possible value) is given by (2).

$$|E|^2 = 1 - \cos\left(\pi * \frac{\text{offset error (pixels)}}{\text{background wavelength (pixels)}}\right) \quad (2)$$

This result indicates that for very low spatial frequency backgrounds the error signal due to an offset of a single pixel will be extremely small, (e.g. 0.03% for clutter with a wavelength of 128 pixels) and will limit the accuracy of the offset optimization. However, for the applications we are studying, the background will be rich in spatial frequencies, and this error signal is quite adequate (e.g. 5.0% for clutter with a wavelength of 10 pixels).

It is important that the model be robust to environment and sensor noise. An advantage of the aggregate error signal used in our algorithm is that temporally uncorrelated noise will be averaged out in the sum over the image. To a first approximation, the variance of the error signal will be lower than the variance of the raw image signal by a factor proportional to the number of pixels. (The true statistics are somewhat more complicated since the error calculated at each pixel is rectified). Hence, random noise in the signal should cause little degradation in the optimization of the offset.

While the moving background is eliminated in the difference image, an object being tracked can take on a complex spatial structure that requires further processing prior to final detection. The purpose of the image enhancement network is to perform some of this processing on the difference image and use the processed result to enhance the retina output. The processing performed by the image enhancement network is application dependent. In our implementation, the image enhancement network performs a low-pass spatial filter on the rectified difference image and multiplies this result with the output of the artificial retina. We envision that further processing will be performed to meet application-specific requirements. For example, a readout that multiplexes and digitizes the analog image followed by digital processing, such as the Automatic Centroid Extractor (ACE) chip [?], will be necessary for a complete real-time tracking system.

Although this processing model is very versatile, there are certain limitations imposed by the general approach and the model in its current form: objects to be tracked should have a different velocity than the background; objects should typically fill only a small portion of the total field of view (FOV); and the velocity of the background must be relatively constant across the FOV. If an object has the same velocity as the background, it cannot be distinguished from the background by its motion. If an object fills a significant part of the FOV, its contribution to the total scene will bias the background motion adaptation. In the worst case, the object fills so much of the FOV that it essentially becomes the background. Finally, if the background velocity is not constant across the entire FOV, the image shift will be misaligned and portions of the background will be visible in the output. We are currently evaluating techniques to overcome these limitations.

### 3 Simulation Results

We wrote a simulator which allowed us to evaluate and explore variations of the retina model. To preserve the analog characteristics of the model and provide the necessary flexibility for variations, we chose to implement an abstraction of an electronic prototyping breadboard. Elements of the model are represented as analog circuit modules which can be "plugged in" and "wired" to other modules on the breadboard. A well-defined interface simplifies the task of writing new circuit modules, and existing modules can be grouped together to provide arbitrarily complex modules.

The image data used in evaluating our circuit designs was derived from a database of real and synthetic imagery. The synthetic data includes various simulated cloud scenes generated with a fractal program, earth scenes generated by a ray-tracing program, and a

frequency modulated (chirped) two-dimensional sinusoid. Our real data is comprised of a sequence of images looking down upon the Earth from the Space Shuttle Challenger during the deployment of the Long Duration Exposure Facility (LDEF) satellite (Shuttle mission STS 41C). A sequence of images representing the output of the photodetector array was produced by spatially sampling a single frame of a database image (interpolation was used to obtain subpixel velocities).

One application of our software simulator in evaluating a given design is to determine the open-loop (no feedback) response to a moving background. Figure 2 illustrates the open-loop response of the design presented in this paper to a cloudy earth background moving at a velocity of  $-0.25$  pixels per delay (this image sequence was derived from the Space Shuttle data). The error feedback to the image-shifting network was disabled, and the weights of the image shifter were initialized to values that yielded the desired spatial offset. A probe was inserted in the circuit to save the error values to a file, and the simulator was run. Next, the average error for the entire run would be computed, and another run would be performed with a different spatial offset. Performing a set of such simulation runs over a range of spatial offsets yields a curve such as that shown in Figure 2. The minima of the curve occurs at the spatial offset resulting in maximum background cancelation (since the background in the delayed image of the example is displaced by  $-0.25$  pixels, a  $+0.25$  pixel offset is required to bring the output of the image shifter in alignment with the output of the artificial retina). For spatially simple backgrounds, the open-loop error curve has a straight "V" shape. The small bends and kinks in the curve of Figure 2 are a result of the complex spatial structure of the clouds in the moving background.

We have also used the simulator to study the behavior of the closed-loop circuit. In this mode, the simulator is simply run on a data sequence, and the spatial shift of the image-shifting network is controlled by the error feedback network. The initial value of the image shifter's weights are all zero (no spatial shift), but quickly begin to change to adapt to the background motion. Similar to the open-loop study, probes are inserted into the circuit in order to save signals and images to files for post-simulation analysis.

An example of the circuit's closed-loop performance using the Space Shuttle data with a superimposed moving object (small Gaussian blob) is shown in Figure 3. The background in the circuit input (Figure 3a) moves  $-0.25$  pixels during the delay period of the background removal network. The object, located above and to the left of the arrow in Figure 3a, moves  $+0.25$  pixels during the same time interval, and has a relative intensity of fifteen percent with respect to the peak-to-peak background intensity variation. Figure 3b demonstrates the circuit's ability to remove the moving background in the input. Although the object is not easily distinguished in the input, it can clearly be seen in the output.

Stability and noise sensitivity are important concerns with systems involving feedback. Our current circuit design contains no damping elements. Appropriate choices of the feedback gain and amplifier cutoff/saturation levels avoid wild instabilities. However, after the system has adapted to the motion of the background, we observed that it tends to fluctuate slightly around the optimal spatial offset value.

In addition to the simulations described above, which were performed in batch mode,

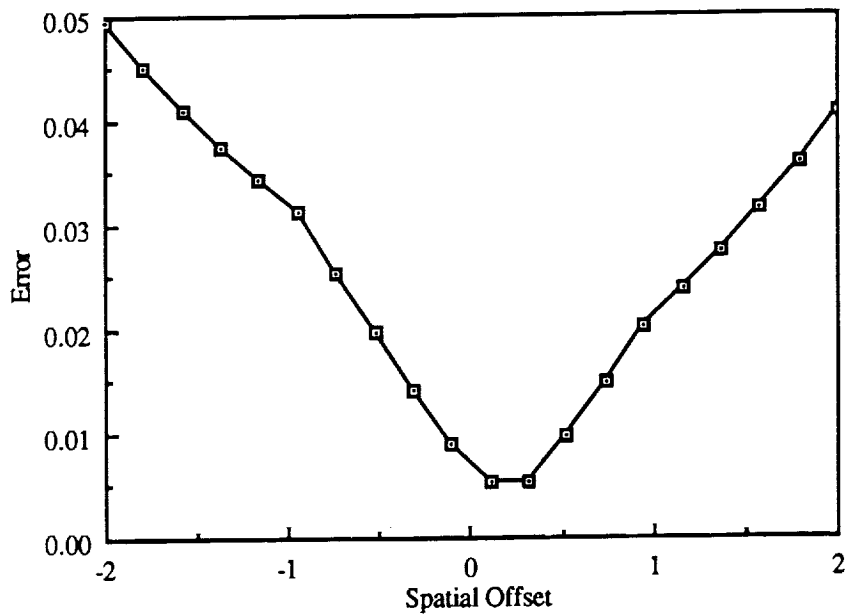


Figure 2: The system open-loop error in response to a background moving at a rate of  $-0.25$  pixels per delay period. As the spatial offset introduced by the image-shifting network approaches the complement of the background displacement, the error between the shifted, delayed image and the unshifted image approaches a minimum. Since the background in this case has a velocity of  $-0.25$  pixels/delay, the optimum spatial offset is  $+0.25$  pixels. The derivative of the error is used to correct the weights in the image-shifting network to minimize the drift error.

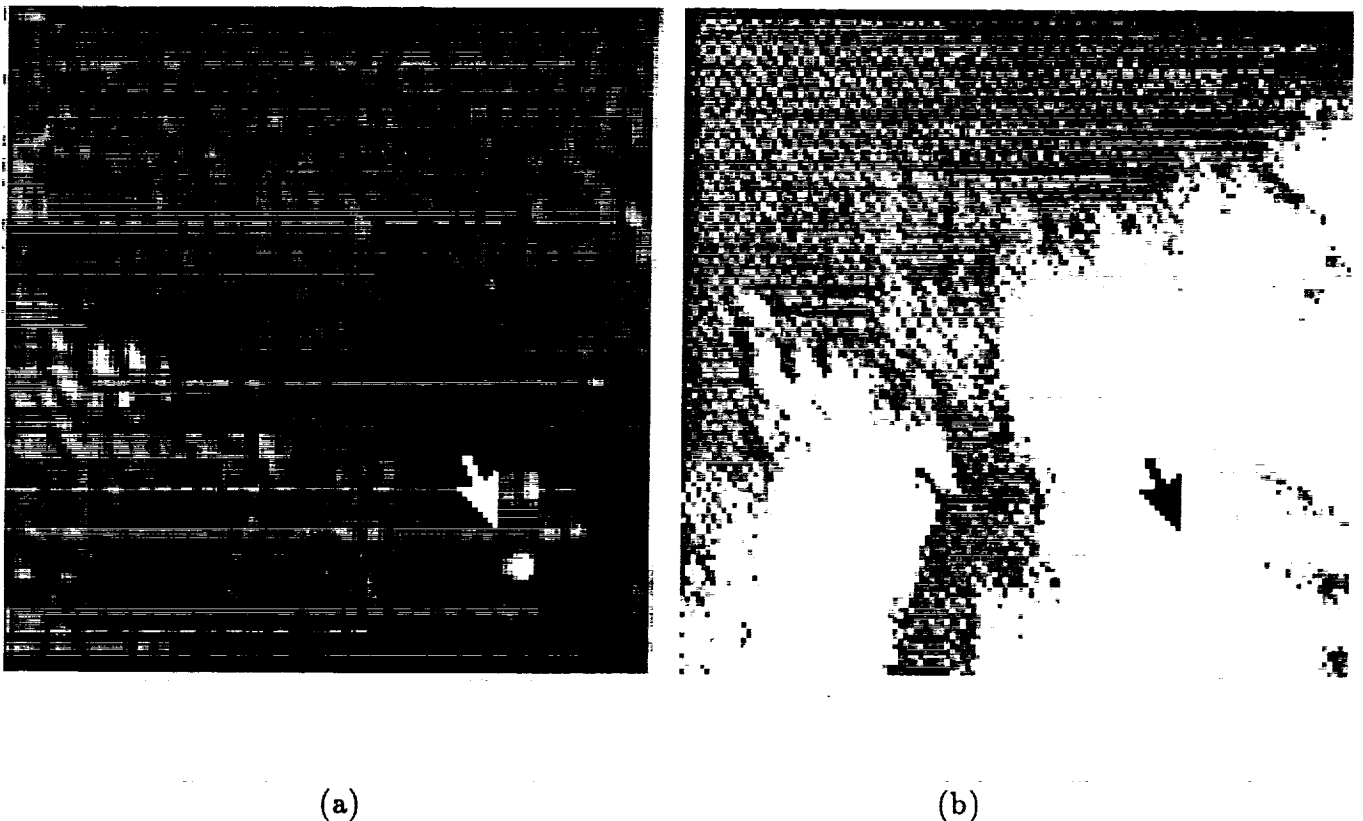


Figure 3: (a) An input image taken from a sequence of data used in computer simulation, and (b) the corresponding enhanced output image. The earth background in the input image is moving at a rate of  $-0.25$  pixels per delay period. A superimposed object moving at a rate of  $+0.25$  pixels per delay period is not easily distinguished in the input image, but can clearly be seen in the upper-left corner of the enhanced output image. The relative intensity of the object with respect to the peak-to-peak background intensity is fifteen percent. Initially, the spatial offset of the image-shifting network is set to zero, and after a few simulation steps, the weights of the image-shifting network adapt to the background motion.

we also implemented a real-time version of the algorithm on a PIPE image processing computer [?]. The PIPE consists of eight processors operating in parallel, each of which can perform complex operations on several frames of data in 1/60th of a second, and an ISMAP board which sums all pixels in a single image. The input to the PIPE implementation came directly from a tripod-mounted video camera. The output was displayed on a video monitor. The scalar error signal, which was computed by the ISMAP board, was passed from the PIPE to an IBM PC where the adaptation algorithm determined the new image-shifting network weights which were then passed back to the PIPE.

This real-time PIPE implementation allowed us to test the performance of the algorithm under real-world conditions (environment noise, sensor noise, and jitter caused by vibrations in the camera), and also determine the dynamic offset adjustment for a wide variety of backgrounds. As demonstrated by a video-tape we made using the PIPE, the algorithm performs well under these conditions.

## 4 Z-Plane Implementation

A typical optical sensor system includes optical elements, a planar array of detectors, and a CCD to multiplex the detector signals into a single signal. In such systems, processing on the focal plane is usually limited to the integration, amplification, and serial readout of the detector signals. Many operations that are currently performed off the focal plane on the digitized detector signals, such as spatial and temporal filtering, would have significantly higher performance if they could be performed in parallel directly on the continuous analog detector signals. Recent advances in fabrication and packaging technology provide the ability to stack analog or digital processing chips together and bond the stack onto the back of a detector array (the "Z" dimension) [?]. Using this technique, hardware that can exploit continuous analog image signals may now be sandwiched between the detector array and readout electronics to form a compact, cube-like image processing device.

The process of manufacturing a Z-plane module consists of thinning integrated circuit (IC) wafers to a desired thickness by precisely grinding the IC substrate, separating the IC wafer into individual circuit dies, laminating the circuit dies into a stack, forming external connections to the laminated circuits, and bonding the stack to the detector array. Z-plane modules with detector array sizes of 128x128 have been achieved, and arrays of up to 256x256 elements are in the range of current Z-plane technology. Larger focal planes have been constructed by tiling the focal plane with Z-plane modules, and a specific Z-plane implementation has been shown to have superior signal-to-noise characteristics, provide more data processing, and consume less power than a comparable CCD implementation [?].

The real-time processing required for our applications currently cannot be achieved on a conventional digital computer; however, our model maps nicely to the parallel, pipelined structure offered by Z-plane technology. Figure 4 illustrates a preliminary Z-plane packaging concept for our processing model. A photodetector array is bonded onto the first layer which implements the image conditioning network. The background removal network is

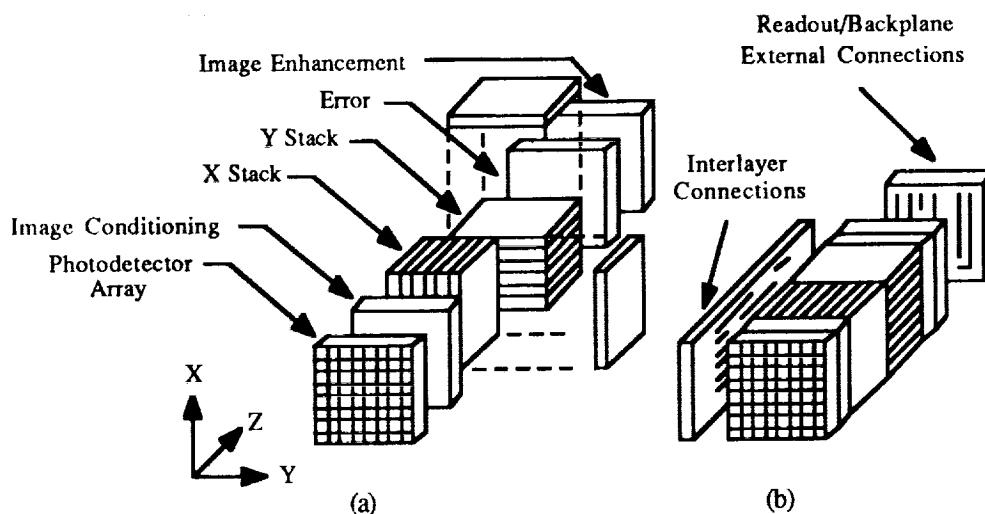


Figure 4: A preliminary Z-plane design of our model. (a) The exploded view shows the distribution of processing modules along the Z axis. The first layer is a photodetector array. An array of amplifiers and a spatial-temporal averaging network that mimics a portion of the outer retina is implemented in the following layer. The background removal network is partitioned among the next three layers (the X and Y processing stacks perform the delay and spatial shift, and the next layer computes the difference and error). Image enhancement is performed in the last processing layer. (b) A partially assembled cube illustrates the use of cube faces for interlayer and external communication.

partitioned into three layers. The first two layers implement the delay and image-shifting network. The first of these layers performs a delay and weighted sum of the inputs in the X direction, and the second layer completes the weighted sum in the Y direction (the stacking of individual IC dies for these two layers are shown in Figure 4a). The third layer of the background removal network implements the difference and error computations, and the last layer performs image enhancement. The faces of the assembled cube provides additional interlayer communication (such as the feedback signals to control the image shifter), and a readout module would be bonded to the back of the cube to provide an external interface. The multiplexed readout from such a device could be either analog or digital, depending on the nature of the readout module. Note that all signals up to the readout module are analog.



## 5 Summary and Future Work

We have presented an analog retina model which adapts to background motion in order to enhance objects with velocities different from that of the background. The results we have obtained from computer simulation demonstrate the model's ability to perform such enhancement for one-dimensional motion. We also presented a hypothetical implementation of our model using "Z" focal plane technology.

Since the one-dimensional simulation results are very promising, we are now proceeding to study the remaining research questions to be answered before creating a more detailed design to be implemented with Z-plane technology. We are currently investigating backgrounds with two-dimensional motion. Although in principle this is a straight forward extension to the current model, the error minimization is now in two-dimensions and may be much more sensitive to system control parameters.

Another important issue is how the system will perform in the presence of external spatial-temporal noise and internally generated noise (such as noise produced from analog component drift or component nonuniformity). Some of the noise will be removed by the artificial retina and by the implicit spatial averaging in the error signal, but we must verify that the remaining noise does not cause the optimization of the spatial offset to become unstable or experience significant drift.

We are also evaluating alternate image enhancement strategies (many of these would naturally be specific to a given application) and techniques to handle significant background velocity differences over the FOV.

## 6 Acknowledgements

We would like to thank the FMC Corporation for providing us access to their PIPE image processing computer. This work has been performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

## References

- [1] J. Carson, "Applications of Advanced "Z" Technology Focal Plane Architecture," Proc. SPIE, vol. 930, pp. 164-182, 1988.
- [2] J. Dowling, *The Retina: An Approachable Part of the Brain*, Belknap Press of Harvard University Press, 1987.
- [3] F. Eeckman, M. Colvin, and T. Axelrod, "A Retina-like Model for Motion Detection," Proc. IJCNN, vol. 2, pp. 247-249, 1989.
- [4] T. Axelrod, T. Tassinari, G. Barnes, and K. Cameron, "A VLSI Centroid Extractor for Real-Time Target Tracking Applications," Proc. SPIE, vol. 1154, pp. 306-312,

4.1.12

1989.

- [5] J. Martin and T. Sangston, "Z-plane Comparison with CCD for Lightning Mapper Application," Proc. SPIE, vol. 1097, pp. 16-27, 1989.
- [6] Randy Luck, *An Overview of the PIPE System*, ASPEX Inc., 536 Broadway St. 10'th Floor, New York, NY.

# Low Power Signal Processing Research at Stanford

J. Burr, P.R. Williamson and A. Peterson

Space, Telecommunications, and Radioscience Laboratory  
Department of Electrical Engineering  
Stanford University  
Stanford, Ca. 94305  
burr@mojave.stanford.edu

**Abstract -** This paper gives an overview of the research being conducted at Stanford University's Space, Telecommunications, and Radioscience Laboratory in the area of low energy computation. It discusses the work we are doing in large scale digital VLSI neural networks, interleaved processor and pipelined memory architectures, energy estimation and optimization, multichip module packaging, and low voltage digital logic.

## 1 Introduction

Our research in low energy computation for signal processing is being supported in large part by NASA. The neural network research is being funded by the Center for Aeronautics and Space Information Sciences (CASIS). Low energy computing research is being funded by NASA grant NAGW1910, "Low power signal processing technology for space flight applications".

## 2 Overall motivation

Our research in low energy computing is driven by the need to maximize computation rates in power constrained environments. Space based data systems and large scale neural networks both require low energy per operation; in flight systems, to minimize power consumption during data gathering, processing, storage, and communication; in neural networks, to achieve the necessary computation rates within manageable power budgets. These systems are characterized by high sustained levels of computational effort, unlike typical portable computer applications, which tend to have bursty, and much more modest, information processing requirements.

## 3 4:2 adder based architectures

We have been building deeply pipelined, parallel signal processors since 1985 [17,18,3,2,19]. We came up with a multiplier architecture which struck a balance between throughput, latch overhead, and regularity [18]. The multiplier consists of a tree of "4:2 adders" (see Fig

### 4.2.2

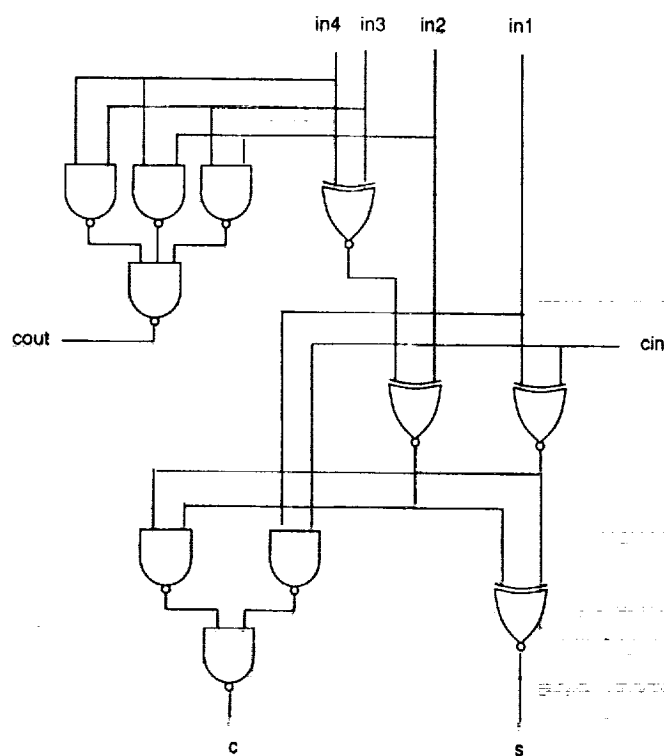


Figure 1: 4:2 adder. The critical path contains three xors in series.

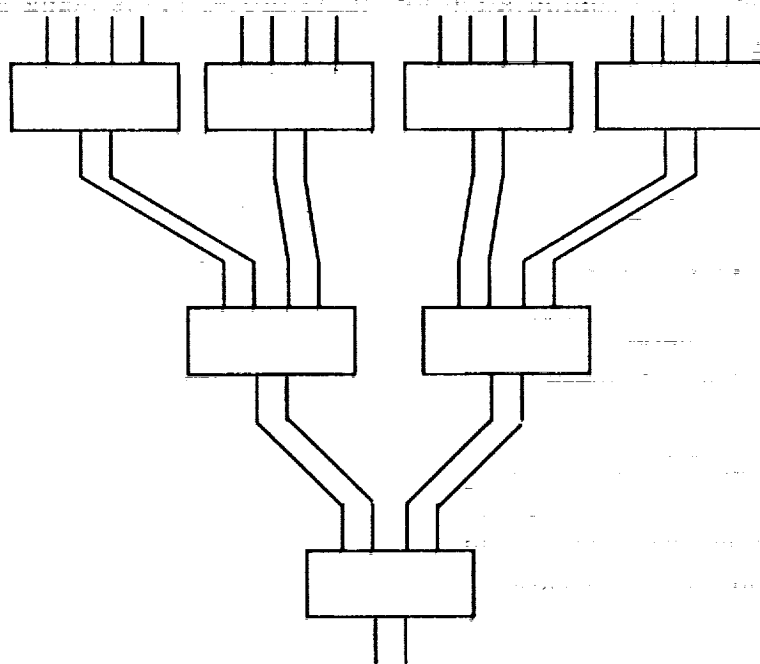


Figure 2: 4:2 multiplier.  $N$  partial products are reduced to 2 in  $\log_2(N)/2$  stages of 4:2 adders.

2). A 4:2 adder (see Fig 1) has 4 inputs, a carry in, and generates two outputs and a carry out. The carry out does not depend on the carry in. The 4:2 adder can be implemented using two full adders, but a direct logic implementation can reduce the critical path from 4 xors in series to three. A multiplier built out of a tree of 4:2 adders has a much more regular structure than a Wallace tree [1], which uses a full adder to reduce three partial products to two at each stage. The 4:2 tree reduces 4 partial products to two at each stage, and has the self-similarity of a binary tree. A 4:2 adder can efficiently accumulate successive products in carry-save form. It can also be used in an ALU to perform arithmetic operations in time independent of the number of bits in the operands.

We have recently shown that power is minimized in a parallel multiplier when  $ld = 11$  [10]. A 4:2 adder has a logic depth of 10, including latches. By comparison, RISC microprocessors typically have logic depths around 40.

We currently have a number of projects which are implementing architectures based on the latency in a 4:2 adder. We were becoming concerned about the feasibility of running systems at the clock rates implied by a logic depth of 10: in 0.8 micron CMOS, a 4:2 adder based clock generator circuit runs at 400MHz [20]. However, similar speeds have been reported elsewhere [21]. Recently, with the opportunity presented by tiled architectures and 3D multichip modules as discussed in [10,27], it appears that deeply pipelined architectures can also achieve good performance at very low energy.

## 4 Neural Nets

Large scale neural nets will require on the order of  $10^{15}$  connections per second (CPS) [9]. Digital VLSI neurochips reported so far require around 1nJ per synaptic connection [22];  $10^{15}$  CPS would require a megawatt! Biological neurons require around 1fJ per synaptic connection, 6 orders of magnitude less. Attaining biological energy efficiency in silicon is a formidable challenge. We have identified a number of factors which together may reduce connection energy by 5 orders of magnitude to 10fJ per connection, permitting  $10^{15}$  CPS at around 10 watts. These include: reduced arithmetic precision (10x), reduced feature size (10x), and low voltage operation (1000x).

In addition to investigating performance of large networks, we are implementing a digital Boltzmann machine [22] to demonstrate the viability of reduced precision, pipelined digital learning machines. The chip is being implemented in 2.0u CMOS, and consists of 32 5-bit neural processors, each supporting 1K 5-bit weights and capable of 80MHz operation. The chip will be capable of 2.5 billion connections per second, and 320 million connection updates per second.

## 5 Pipelined Memory

We are implementing a pipelined memory architecture (see Fig 3) which achieves high throughput by recursively subdividing the memory array into sections which can be traversed in a single cycle. Addresses are partially decoded in each section. The remaining

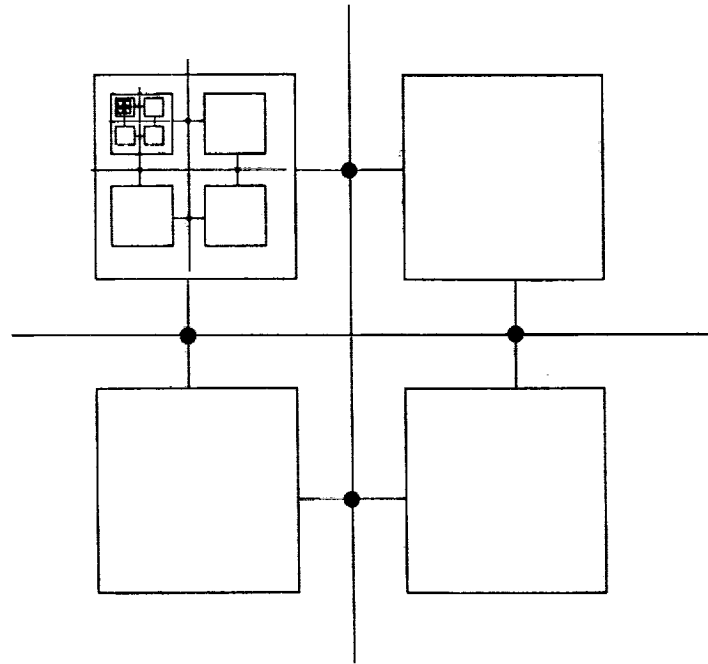


Figure 3: Pipelined memory

address bits are routed to the appropriate subsection where additional bits are decoded. At the lowest level, the remaining bits are decoded and data is read out of or written into a memory block. For read operations, the data is delivered back up through the subsections on subsequent cycles until it emerges at the pads. For write operations, the data accompanies the address down the tree.

The size of the memory block is matched to the propagation delay through a 4:2 adder. This turns out to be about 32 words x 32 bits. We have written an optimizer which sizes the transistors in this block for the minimum area and power that matches the delay [25]. We pipeline the address decode and data return, placing pipestages to minimize power dissipation. Power dissipation in the memory is greatly reduced by selectively clocking the portion of the memory which contains the data, leaving the rest of the system on standby.

Hierarchical memory organization first appeared in Mead and Conway [12,11], but this architecture was not pipelined. An unpipelined binary tree memory was described at the 1987 International Test Conference [8,26]. Hierarchical address decoding was reported in a 4Mb SRAM with selective enable to reduce power dissipation [7].

A pipelined memory architecture was discussed in [28]. The CT7C158 is a pipelined 64K SRAM offered by Cypress Semiconductor, who say: "Pipelined RAMs are used in writeable control store, DSP and logic analyzer/tester applications where throughput is the critical parameter."

Our pipelined memory is the first to combine hierarchical address decoding and selective clocking to maintain very high throughputs and very low power dissipation.

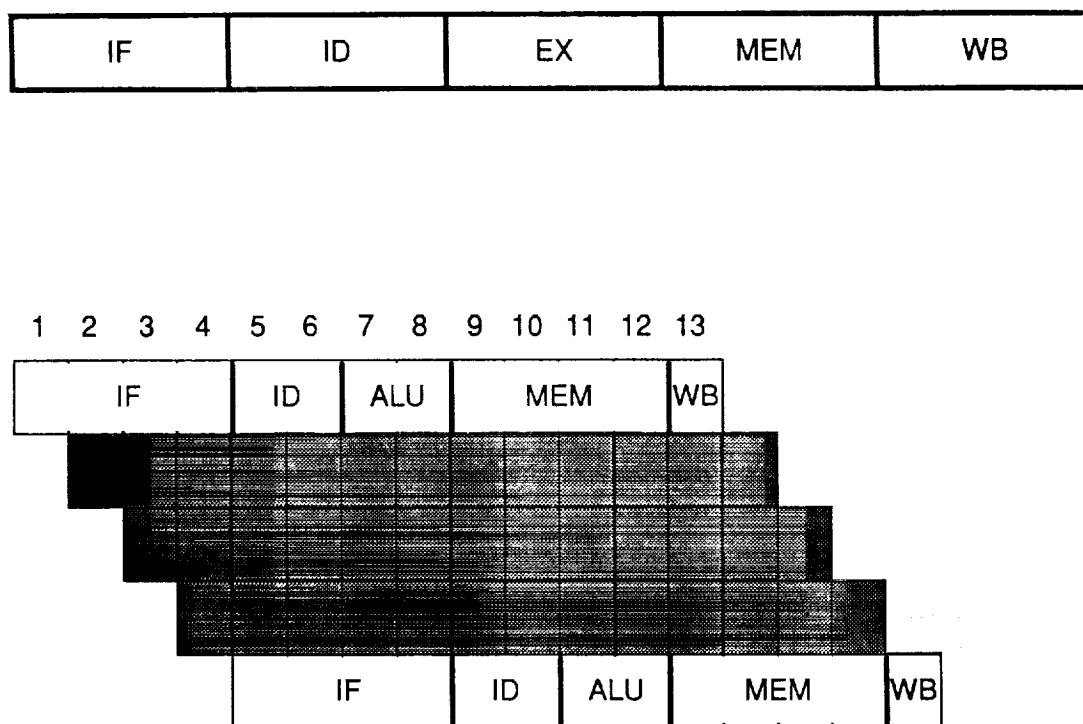


Figure 4: Interleaved processor pipeline, with a normal RISC pipeline for comparison. In this example, instruction fetches and memory accesses take four cycles. There are four independent instruction streams in various passes of execution.

## 6 Interleaved Processor

We are working on a processor architecture which achieves high performance by interleaving independent instruction streams on a deeply pipelined processor (see Fig 4). The number of independent streams is matched to the latency in the pipelined memory. The clock frequency is a multiple of a RISC clock, and is obtained by placing extra pipestages at critical points in a RISC architecture. The number of extra pipestages is smaller than expected because many of the normal RISC stages do not use up an entire clock cycle. Our objective is to achieve a 4x speedup over RISC in a given technology, and to implement a subset of the MIPS R3000 instruction set. We are experimenting with a variety of power reduction techniques at the circuit and system level in the processor design.

Multiple instruction stream processors have been built before (Burton Smith's work on HEP, Horizon, and Tera [16,15]), but only in the context of large supercomputers and not single integrated circuits, and not matched to the latency of a pipelined memory. Edward Lee at UC Berkeley proposed an interleaved architecture for use in signal processing [14], but his design is not pipelined as deeply as ours, and does not include pipelined memory. The only reference we have found so far which describes an interleaved processor and a pipelined memory is a Japanese paper on gate-level pipelined Josephson Junction circuits [28], which also describes a method to increase the throughput of CMOS memory

#### 4.2.6

by pipelining, but the two concepts are not synergized, and the memory organization is not discussed. Stone and Cocke say "some combination of long pipelines and multiple interleaved instruction streams may eventually prove effective for combining high speed and high efficiency" but give no details [23].

The RISC community is also investigating techniques for increasing performance. The two chief techniques are superscalar and superpipelined architectures [5]. In superscalar, more than one instruction may be in progress at a given time. In superpipelined, the RISC pipe is broken into a number of smaller stages with reduced logic depth. Both of these approaches result in added control complexity managing the potential hazards and resource conflicts which may result.

Superscalar machines, such as the Intel I860, fetch more than one instruction on each cycle, and execute in parallel whenever possible. There are restrictions in the combination of instructions which can be issued simultaneously. Superscalar increases resource utilization but does not increase the throughput of a given functional unit.

We reduce RISC logic depth by a factor of 4, and introduce 4 independent, interleaved instruction streams. The streams are kept independent to avoid the hardware complexities associated with managing a highly pipelined single thread of control. Each instruction stream executes its next instruction every fourth cycle. The control complexity is no worse than for a RISC machine but the throughput is 4 times greater on problems that can be parallelized. Fortunately, these are commonplace in signal processing. The architecture also supports zero-overhead context switching of up to 4 processes. This is very useful in embedded real time control applications.

### 6.1 Timing

Real time signal processing tasks often require "precise" timing. This is not easy in cache-based architectures, since cache miss recovery times can often be data dependent. The pipelined memory/interleaved processor behavior is precise: instruction latencies are fixed. Memory fetches always takes 4 cycles. There are never any cache misses. Branch timing is precise.

In a conventional RISC machine, the latency that takes place during a branch is unpredictable, because it depends on whether the target address is in the instruction cache, and if so, how it is aligned within the cache entry that contains it. Given a 4 cycle latency to fill a line in the cache, and a cache linewidth of 4 words, a branch target will only point to the first word in the line 25% of the time. The system must stall fetching the line following the line containing the branch target address. The AMD29000 "branch target cache" solves this problem by aligning cache lines to branch targets. This increases the complexity of the memory subsystem. The interleaved processor solves this problem by maintaining a fixed latency on every instruction fetch.



## 6.2 Energy

Our objective has been to maximize overall performance. With the advent of Multichip module technology, the performance of an individual chip must be considered in light of the system. We now are designing to maximize performance at minimum energy. The best way to do this is to obtain the maximum possible throughput, and use the performance margin to lower the supply voltage until all the available area is used and the power budget is met.

The clock frequency can be increased by a factor of 4, so that each stream can execute as fast as a RISC processor in the same technology, and the processor can achieve 4 times RISC performance. This implies 400MHz in 0.8 CMOS. Although this is feasible for small numbers of processors, we plan instead to lower voltage by a factor of 4, to 1.25V. This will give us the same 2D performance density as a RISC machine, but will require only 1/16 the energy per operation and 1/64 the power. We can capitalize on MCM technology to achieve 64 times the performance with 64 times the area for the same power budget.

Also, because resources are pipelined, more time is available to wake up an idle resource or put it on standby. Resources only need to be clocked if they are being used. If a resource is used by one stream, but not by the next, the inputs to that resource can retain their previous values.

Register files normally consume a significant portion of the power budget. Since each stream has its own register file, the access rate to a register file can be 1/4 the system clock frequency. Conventional SRAM is faster and lower power than multiported register files since the bitlines never have to swing more than 100mV for reading or writing. If the SRAM can be accessed in a single cycle, it can emulate a 4-port memory which can support any combination of up to 4 reads or writes every 4 cycles. In its standard configuration it would be accessed sequentially to fetch two operands and write back a third. Whether this results in less energy depends on how often operand addresses are repeated on successive instructions.

## 6.3 Area

The interleaved processor should require area comparable to a RISC processor because four sets of registers, program counters, and other state registers take no more area than on-chip instruction and data caches.

## 7 Multichip Modules

Multichip module packaging provides a number of significant new opportunities in system architecture and implementation. Bare die can be placed much closer together than packaged parts, leading to shorter wires and reduced communication energy. Area bonding reduces lead inductance, permitting higher frequency interchip communication. Small bonding pads and high connective capacity support seamless interchip communications

optimized for propagating signals a few centimeters. Intrinsic bypass capacitance due to thin dielectric separation of Vdd and Gnd planes results in higher noise immunity.

The net result is the opportunity to reduce communication energy and increase system level performance by orders of magnitude compared to conventional packaging techniques. We are developing interconnect structures, data transmission circuits, and clock distribution structures for high performance (hundreds of MHz), low power (tens of mW) MCM systems. Much of our work in this area has been reported in [24].

We have designed a test module which is being fabricated by ATT. It includes passive structures for measuring capacitance, crosstalk, and characteristic impedance of a variety of conductor geometries. It also has two sites for MOSIS TinyChips which will test the interconnect by exchanging pseudorandom bitstreams through single ended and differential transceivers at data rates in excess of 200 MHz.

## 7.1 Tiled architectures for signal processing

The opportunity exists to extend the concept of regularity and locality so widely used in VLSI design to the multichip module level, and to identify a set of processor tiles which can tessellate the plane to generate massively parallel architectures. We are investigating a variety of "tiled" architecture opportunities. We have extended our neural net Boltzmann machine architecture to accommodate an arbitrarily large two dimensional array of chips.

# 8 Multiprocessing

The interleaved processor is inherently a symmetric shared memory multiprocessor. Memory consistency is guaranteed because there is no cache. We are investigating ways to interconnect interleaved processors for massively parallel multiprocessing.

## 8.1 Hierarchical pipelined ringbus

One possible organization of a massively parallel system is a "hierarchical ring bus" architecture which supports high bandwidth pipelined data exchange among multiple processors. The overall topology consists of rings of processors connected by gateways. Each local ring can sustain data transfers at the processor clock rate. Because the bus itself is pipelined, multiple transactions can be in progress concurrently, up to the number of processors in the ring. One of the nodes in the ring can be a gateway to another ring and can sustain the same I/O bandwidth. We plan to match the bus clock frequency to the latency of a 4:2 adder.

This architecture has been proposed elsewhere [15]. We think it is well matched to the performance and latency of the interleaved processors and multichip module based multiprocessors. In the spirit of interleaved instruction streams, the latency to complete a single bus transaction will be at least equal to the number of processors in the ring, but a separate bus transaction can be in progress simultaneously on each segment of the ring. This will result in substantially higher throughput than conventional bus architectures - in

excess of 1 Gbyte/sec. This architecture is well suited to datastream oriented algorithms common in real time signal processing.

Although this approach introduces single point failures at each node in the ring, when placed in the context of 3D multichip module implementation we think the approach has some significant advantages.

The ringbus concept can be extended gracefully to large numbers of processors by recursively adding subrings connected by gateways. We will be analyzing the implementation complexity, energy, and performance of this approach in comparison to other processor communication networks.

Of key interest is mapping numerically intensive signal processing problems onto this architecture. A 1024 processor system might consist of 64 rings with 16 processors in each ring. At 400 MIPs per node and 1 Gbyte/sec per ring, total performance would be 400GIPS; total throughput would be 64 Gbytes/sec. Ring size can be optimized to balance instruction and communication bandwidth.

## 9 Energy estimation and optimization

We estimate energy using

$$\begin{aligned} E_{ac} &= \frac{1}{2}aCV^2 \\ E_{dc} &= I_{dc}V/f \end{aligned}$$

where  $a$  is the activity ratio, the fraction of transistors switching on each cycle,  $C$  is the capacitance being switched,  $V$  is the supply voltage,  $I_{dc}$  is the DC current, and  $f$  is the clock frequency.

This technique relies on short circuit current being a small fraction of the total.

We are investigating techniques for minimizing power dissipation by minimizing transistor sizes while minimizing short circuit current. These are conflicting constraints, and can lead to substantial power reductions over techniques which ignore short circuit current and assume minimum size devices result in minimum power.

We have modified our timing simulator to measure AC power dissipation by accumulating dumped charge. Preliminary results suggest good agreement with power measurements on fabricated chips. We are extending this technique to measure peak power. We have developed a memory block optimizer which sizes transistors in the pipelined memory to maximize a "merit" function which is a weighted combination of performance, power, and area. We are including the effects of short circuit current on both our transistor sizer and our memory block optimizer.

We have found that transistor sizing is important in optimizing highly pipelined designs. Balancing clock delays is especially important to minimize clock skew in the system. Transistors can also be sized to minimize energy, which involves balancing short circuit current against gate capacitance.

## 10 Low Voltage Digital Logic

Massively parallel architectures tiled on 3D stacked multichip modules can quickly exceed the ability to extract heat from the structure. Reducing the supply voltage promises substantial reductions in energy and power; we are investigating the practical limits to low voltage operation. This area is covered in depth in [10].

Our approach to low energy computation has attracted interest from a number of sources. More detailed investigation into the opportunity is being funded as a "Research thrust" by Stanford's Center for Integrated Systems. These research thrusts involve interaction with technical liaisons from CIS industrial partners. So far, the Ultra Low Power thrust has liaisons at DEC, GE, IBM, Intel, National Semiconductor, and TI.

## 11 Personnel

Who the group is:

Professor Allen M. Peterson, Principal Investigator

P. Roger Williamson, Senior Research Associate

James B. Burr, Senior Research Engineer

### Low Energy Computing

Bevan Baas	computer architecture
Jim Burnham	high speed interconnect
Ely Tsern	interleaved algorithms
Gerard Yeh	Low energy VLSI circuits
Sabeer Bhatia	Low energy process design

### Neural Networks

Kan Boonyanit	Approximate Gradient Descent
Karen Huyser	Wafer Defect Classification
Michael Leung	Texture Recognition
Michael Murray	Precision, Learning, and VLSI

### Collaboration

ATT,	multichip modules
Sun,	energy optimization
Intel,	digital neural network architectures
Ricoh,	neural net coprocessors

## 12 Conclusion

Our research in low energy computation has been motivated by recent trends in VLSI technology, multichip module packaging, and application architectures. We believe the opportunity exists to achieve very high computation rates in power constrained environments by reducing decision, storage, and communication energy.

## 13 Acknowledgements

This research was supported in part by NASA grants NAGW1910 and NAGW419, by a gift from Intel Corporation, and by a grant from Stanford's Center for Integrated Systems. Multichip modules were provided by ATT, workstations by Sun Microsystems, and VLSI fabrication by MOSIS.

## References

- [1] Shlomo Waser and Michael J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, CBS College Publishing, 1982.
- [2] Weiping Li and James B. Burr and Allen M. Peterson, "A fully parallel VLSI implementation of distributed arithmetic", *IEEE International Symposium on Circuits and Systems*, June, 1988, 1511-1515.
- [3] Weiping Li, "The Block Z transform and applications to digital signal processing using distributed arithmetic and the Modified Fermat Number transform", 1988.
- [4] Weiping Li and James B. Burr, "An 80 MHz Multiply Accumulator", PhD thesis, Stanford University, 1987.
- [5] John L. Hennessy and Norman F. Jouppi, "Computer technology and architecture: An evolving interaction", *IEEE Computer Magazine*, 9, 1991, 18-29.
- [6] James B. Burr and James R. Burnham and Allen M. Peterson, "System-wide energy optimization in the MCM environment", *IEEE Multichip Module Workshop*, 1991, 66-83.
- [7] Toshihiko Hirose, Hirotada Kuriyama, Shuji Murakami, Kojiro Yuzuriha, Takao Mukai, Kazuhito Tsutsumi, Yasumasa Nishimura, Yoshio Kohno and Kenji Anami, "A 20ns 4Mb CMOS SRAM with hierarchical word decoding architecture", *IEEE International Symposium on Circuits and Systems*, 1990, 132-133.
- [8] Najmi T. Jarwala and D. E. Pradhan, "An easily testable architecture for multi-megabit RAMs", *IEEE Test Conference*, 1987, 750-758.

- [9] Carol Weiszmann, " DARPA Neural Network Study ", October 1987 - February 1988, *AFCEA International Press*, 1988.
- [10] James B. Burr and Allen M. Peterson, " Ultra Low Power CMOS Technology ", *NASA VLSI Design Symposium*, 1991.
- [11] Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
- [12] Carver A. Mead and Martin Rem, " Cost and performance of VLSI computing structures ", *IEEE Transactions of Electron Devices*, April, 1979, 533-540.
- [13] Kentaro Shimizu, Eiichi Goto and Shuichi Ichikawa, " CPC (Cyclic Pipeline Computer) - an architecture suited for Josephson and Pipelined-Memory machines ", *IEEE Transactions on Computers*, Volume 38, Number 6, June, 1989, 825-832.
- [14] Edward A. Lee and David G. Messerschmitt, " Pipeline interleaved programmable DSP's: Architecture ", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Sept, 1987, 1320-1333.
- [15] Burton J. Smith, " The Horizon Supercomputer ", *Supercomputing*, Oct, 1988.
- [16] Burton J. Smith, " Architecture and applications of the HEP multiprocessor computer system ", *SPIE, Real-Time Signal Processing IV*, 1981, 241-248.
- [17] James B. Burr and others, " A 20 MHz Prime Factor DFT Processor ", Stanford University, Sept, 1987.
- [18] Weiping Li and James B. Burr, " An 80 MHz Multiply Accumulator ", technical report, Stanford University, Sept, 1987.
- [19] Alfred J. Eiblmeier, " A reduced coefficient FFT butterfly processor ", technical report, Stanford University, Oct, 1988.
- [20] Mark R. Santoro, " Design and Clocking of VLSI Multipliers ", PhD thesis, Stanford University, 1989.
- [21] Y. Jiren, I. Karlsson and C. Svensson, " A true single phase clock dynamic CMOS circuit technique ", *IEEE Journal of Solid-State Circuits*, 1987, Volume SC-22, 899-901.
- [22] James B. Burr, " Digital Neural Network Implementations ", *Neural Networks: Concepts, Applications, and Implementations, Volume 2*, Prentice Hall, 1991.
- [23] Harold S. Stone and John Cocke, " Computer architecture in the 1990s ", *IEEE Computer Magazine*, Sept 1991, 30-38.
- [24] James B. Burr, James R. Burnham and Allen M. Peterson, " System-wide energy optimization in the MCM environment ", *IEEE Multichip Module Workshop*, 1991, 66-83.

- [25] Bevan Baas, " A pipelined memory system for an interleaved processor", technical report, Stanford University, Sept, 1991.
- [26] Dhiraj K. Pradhan and Nirmala R. Kamath, " RTRAM: Reconfigurable and testable multi-bit RAM design ", *IEEE International Test Conference*, 1988, 263-278.
- [27] James B. Burr and Allen M. Peterson, " Energy considerations in multichip-module based multiprocessors ", *IEEE International Conference on Computer Design*, 1991.
- [28] Kentaro Shimizu, Eiichi Goto and Shuichi Ichikawa, " CPC (Cyclic Pipeline Computer) - an architecture suited for Josephson and Pipelined-Memory machines ", *IEEE Transactions on Computers*, Volume 38, Number 6, June, 1989, 825-832.





## Technology, Design, Simulation, and Evaluation for SEP-Hardened Circuits

J. R. Adams, D. Allred, M. Barry, and P. Rudeck,  
R. Woodruff, J. Hoekstra, and H. Gardner  
United Technologies Microelectronics Center  
1575 Garden of the Gods Road  
Colorado Springs, CO 80907

**Abstract-** This paper describes the technology, design, simulation, and evaluation for improvement of the SEP hardness of gate-array and SRAM cells. Through the use of design and processing techniques, it is possible to achieve an SEP error rate less than  $1.0\text{E-}10$  errors/bit-day for a 90worst-case geosynchronous orbit environment.

### 1 Single Event Upset

Single Event Phenomenon (SEP) occurs when a particle or heavy ion interacts with the silicon, depositing charge on critical circuit nodes, causing data loss. Devices that retain data (RAMs and flip-flops, for example) are subject to SEP, and a particle interaction in one part of the chip can cause data loss in a circuit located far away. Non-storage nodes can propagate the pulse to other circuit nodes, but no permanent data will be lost. Storage devices like RAMs, latches, and flip-flops may detect false clock pulses, or reset signals caused by a pulse on a circuit node somewhere in the clock, or reset generation and buffering circuitry, and lose data.

The storage node critical charge,  $Q_c$ , is the amount of charge that must be deposited on the storage node in order to upset the stored data. Increasing the critical charge for the sensitive nodes in a cell will decrease the SEP error rate by lowering the probability of encountering an ion with sufficient LET value to upset the cell. This critical charge can be increased by changing the transistor sizes to make the cell more stable, adding parallel paths in the cell, and increasing the feedback switching time. A more stable storage cell will require a greater voltage change on the storage node, or a longer voltage pulse to disturb the data.

### 2 Charge Deposition Model

In the past, the literature has implied that the charge generated by a single event may be modeled by using an ideal current source with an exponential time decay. Using this method, the current source is connected directly to the sensitive node, and the charge applied is the time integral of the current pulse. This method can over-predict the value of the critical charge for the memory cell because the current source causes the junction to which the current source is applied to become forward biased and sink a significant

amount of charge to the power supply. Physically, the charge collection process is self-limiting, and the junction will never conduct current in the forward direction. This is a very common error which has led to substantially over-predicting the SEP hardness of a particular design. This can lead to substantially higher SEP error rates in the circuit than can be tolerated by the system design. In this paper, we describe process, simulation, and design techniques that may be used to improve the SEP hardness of CMOS circuits.

### 3 SEP Simulation

We have used both HSPICE and DAVINCI (tm) [1] to model SEP phenomenon. In the SPICE simulation, the cell sub-circuit is accurately modeled using device parameters which have been demonstrated to correlate with measured silicon values. Figure 1 shows the sub-circuits that model the charge deposition. They use idealized MOS devices as switches, and standard SPICE components for everything else. This modeling technique prevents forward-biasing of the junction and provides a more accurate simulation of the charge required for circuit upset [2]. An internal node, Ndep, is initialized to a voltage, Vdep, by switch N3 where  $V_{dep} \times C_{dep} = Q_{dep}$  ( $Q_{dep}$  is the simulated deposited charge). A switch, P2 or N2, is turned on in approximately 0.1 ns, shorting the node under test to a power supply through a resistor that represents the resistance of the bulk silicon.

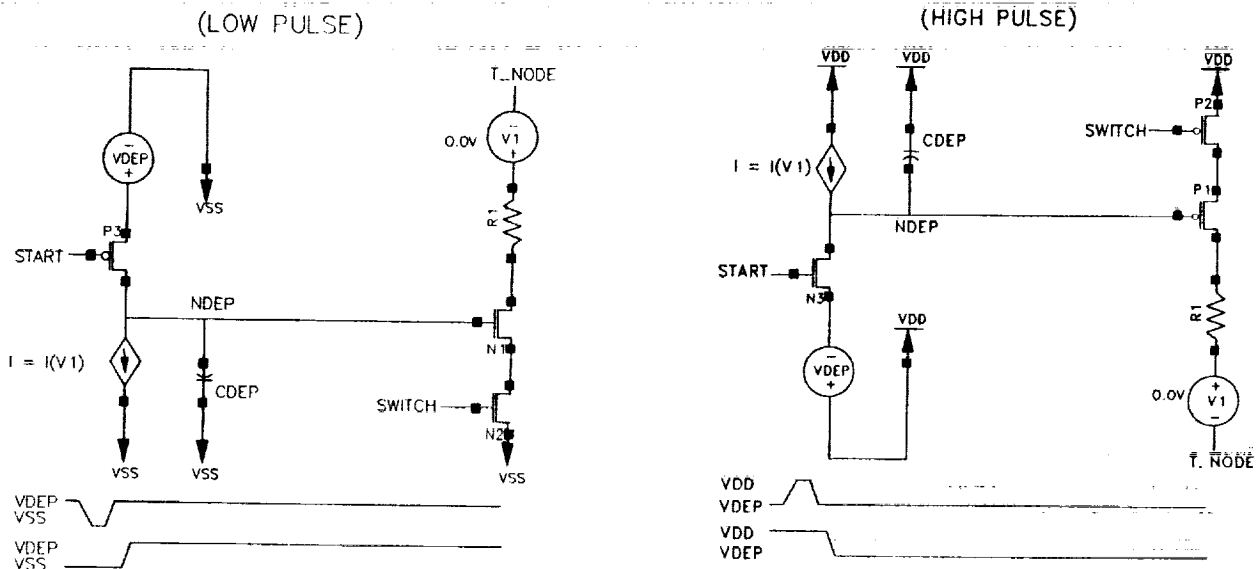


Figure 1: SPICE Charge Deposition Circuit

As charges are added to the node under test, the same amount of charge is removed from the internal node Ndep. When the internal node Ndep is depleted of charge, a switch, P1 or N1, turns off so the circuit can recover its node voltages. A similar but slightly different version of this charge deposition model is used for p+/n- and n-/p+ junction interactions. This technique has proven extremely valuable to optimize the design and layout of logic and memory circuits for SEP hardness.

Numerical simulation of the charge deposition from a heavy ion hit has been attempted by others. [3,4] We have performed three-dimensional numerical simulation of single event upset using the three-dimensional simulator, DAVINCI tm. The simulations were performed on an n-channel junction of a twin-tub CMOS device with approximately 4  $\mu\text{m}$  of an epitaxial layer. In figure 2, the potential contours of the device junction are shown 48 ps after a single event hit with a gold ion. Note that the n+ junction is no longer at five volts and that the distribution of the funnel favors the epi/p-well junction. As the voltage at the n+ junction is reduced, there is less potential difference to the funnel as opposed to the potential difference of the funnel to the epi, which is held at five volts.

seu7gold80\_let Potential 48.0ps

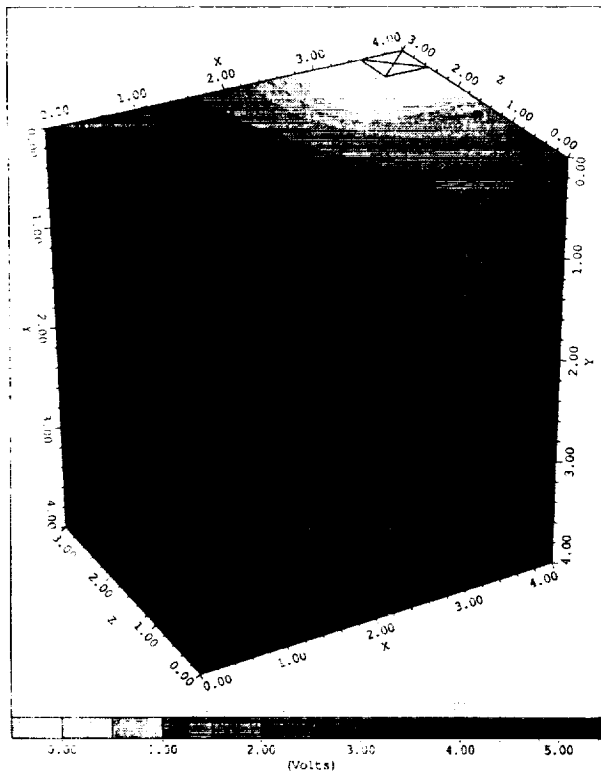


Figure 2: Junction Potential After Ion Strike

seu7- Node Voltages

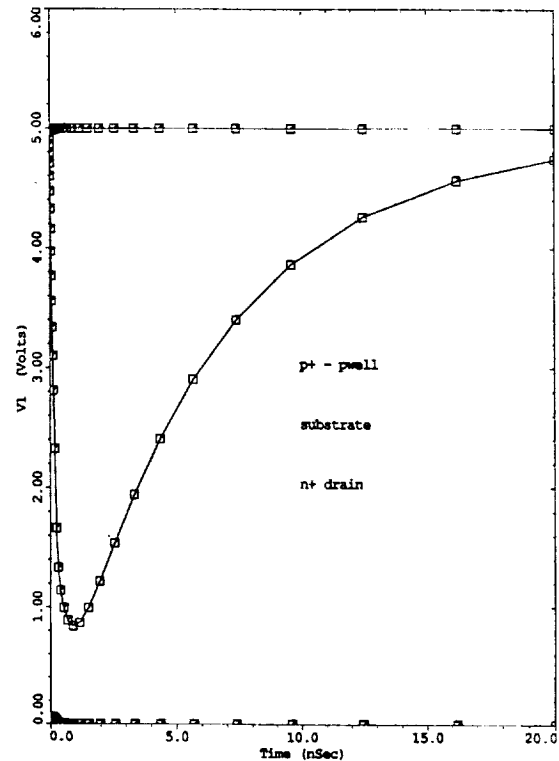


Figure 3: n+ junction vs. p-well and substrate

Figure 3 shows the potential of the n+ junction relative to the p-well contact and epi substrate contact. Note that the time constant is  $\sim 5\text{-}10$  ps. Whether a flip-flop will change state as a result of this SEP event will depend upon how long the n+ junction potential is below the switch point of the cell. The n+ junction can go to zero volts and not cause an upset if it recovers before the zero can propagate back through the cross-coupled logic. We have found DAVINCI tm useful for looking at wafer fabrication process methods for SEP hardening of CMOS devices, as it enables the evaluation of effects of doping concentration, epi thickness, etc.

## 4 Heavy Ion Testing And SEP Numerical Calculation

Heavy ion testing for this work was performed at Brookhaven National Laboratories, using their Tandem Van de Graaff system. Three ion conditions were chosen for this testing. They included Gold at 350 MeV, Iodine at 320 MeV and Bromine at 285 MeV. The LET values were further varied by adjusting the angle of incidence from 0 to 60 degrees.

To determine the SEP error rate from measured data, the effective cross-section is selected at an LET of  $100 \text{ MeV} \times \text{cm}^2/\text{mg}$  (surface value). The upset rate calculation can be performed using either CREME [5] or SpaceRad [6] programs. The simulation conditions for determining the error rates quoted in this paper are as follows:

1. Geosynchronous circular orbit, 35900 km;
2. Orbital inclination, 0 degrees;
3. Adams 90% worst-case environment, including the earth's shadow and geomagnetic storms;
4. All ions from Hydrogen through Uranium  $1 < Z < 92$ .

## 5 Process Techniques For SEP Hardening Of CMOS Devices

Wafer fabrication processing can have a strong effect on the SEP sensitivity of CMOS circuits. As shown in the DAVINCI tm simulations above, over half of the charge deposited by a heavy ion can be collected at the epitaxial junction, away from sensitive circuit nodes, if the epitaxial layer is sufficiently thin. Also, because the drive current of p-channel devices is typically less than that of n-channel devices, the most SEP-sensitive nodes tend to be n-channel nodes supported by p-channel transistors. Therefore, to minimize the charge collection on these nodes, a p-well type process is desirable since the p-well-to-substrate junction will help to collect a substantial portion of the deposited charge.

Also, because n-type dopants (n-type substrates are used for a p-well process) diffuse much slower than p-type dopants, it is possible to fabricate much thinner epitaxial layers for a p-well process, further improving the SEP sensitivity of the technology. The lower sheet resistances and (typically) higher doping of a p-well process also help to eliminate SEP-induced latch-up. High doping concentrations also help to increase the junction capacitance, further improving SEP susceptibility. Thin gate oxide also increases node capacitance, thereby increasing the critical charge on a node and improving SEP hardness. Poly-resistors or natural p-channel transistors can also be added to the process to allow the design of high-density SEP-hardened memory cells.

SOS and SOI (Silicon-On-Sapphire and Silicon-On-Insulator) processes reduce the amount of charge collected on the junction and the effective critical charge on each node. Thin-film SOI devices are also sensitive to bipolar snap-back. This has the effect of making

the channel region of the n-channel transistors sensitive to SEP upset. Therefore it is not sufficient to process a design on SOS/SOI substrates to obtain good SEP performance. Design and special processing techniques must be used also to assure SEP hardness of the circuits.

Commercial CMOS wafer fabrication processes usually do not consider SEP upset and latchup in their design. They are optimized for speed and density, both of which can compromise good SEP performance. UTM C has designed its twin-tub epitaxial p-well CMOS process (including poly-resistors) and layout rules to provide an optimum balance between good SEP performance, latchup immunity, speed, and density.

## 6 SEP Hardening Techniques For Memory Circuits

In the design of memory systems, several techniques may be used to provide SEP-insensitive memory systems. These include the use of redundant memory with voting logic and/or error detection and correction. Both of these techniques require additional system overhead and result in a degradation of system performance, as well as increased cost and weight. Some systems cannot afford this additional overhead, and therefore require the use of SEP-hard SRAMs.

Several SEP hardening techniques for SRAM memory cells have been reported in the literature. These include the use of cross-coupled resistors, cross-coupled capacitors, and cross-coupled p-channel transistors. [7,8] A schematic diagram of a memory cell with cross-coupled resistors (as used in UTM C's rad-hard 64K SRAM) is shown in figure 4. All of these techniques serve to increase the write time constant of the cell, thereby increasing the effective critical charge on the internal nodes of the cell. All of the techniques increase the wafer fabrication processing complexity and the area of the SRAM cell. The advantages and disadvantages of each are shown in Table 1.

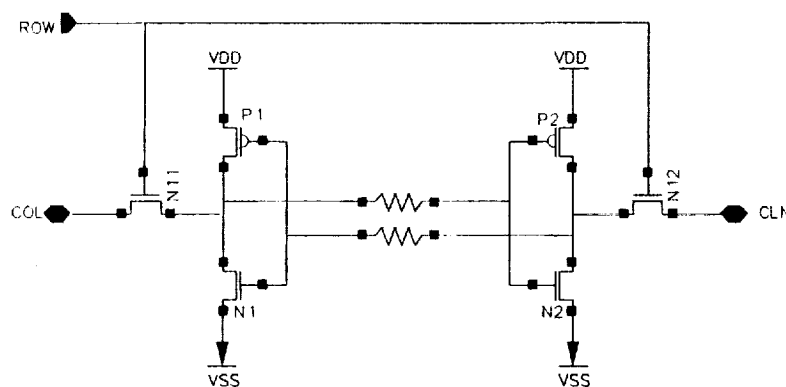


Figure 4: Memory Cell Schematic

Today, cross-coupled resistors are used in many SEP-hard designs primarily because the processing required to add and control the resistors is a relatively straight forward extension of standard CMOS SRAM processing techniques. However, if proper care is not taken to optimize the cell layout for performance, SEP sensitivity, and resistor tolerance,

SRAM Cell Hardening Technique	Advantages	Disadvantages
Cross-coupled resistors	Known processing. First published technique.	TCR of resistor causes SEP sensitivity to change over temperature. Cell area. Effects of resistor geometry.
Cross-coupled capacitors	Minimum variation of SEU with temperature.	Cap oxide defect sensitivity. Cell area.
Cross-coupled transistors	Minimum cell area.*	Process control on transistor $V_t$ .

\* Technology and layout dependent

Table 1: Advantages and Disadvantages of SRAM SEP-Hardening Techniques

SEP performance may be greatly degraded at high temperature (above 85 C), and write cycle time may be degraded at low temperature (-55 C).

## 6.1 SEP-Hardened Memory Design

We have taken great care to optimize the performance of the rad-hard 64K SRAM over the entire military temperature range (-55 C to 125 C). The size of the transistors in the cell were optimized in concert with the cross-coupled resistor values using the SPICE simulation techniques described earlier to assure that the product would be manufacturable and meet data sheet specifications, including a 1E-10 error/bit/day requirement, over the entire military temperature range. UTMIC's 64K SRAM memory cell is more stable than most SRAM memory cells because the p-channel transistor in the 64K SRAM memory cell is larger than the n-channel transistor and will supply almost as much current. The increased p-channel size increases the switch point of the cross-coupled inverters from approximately  $V_{tn}$  (0.8 V) to approximately  $V_{dd}/2$  (2.5 V). The increased switch point requires that a particle-induced voltage pulse on one of the storage nodes exceeds  $V_{dd}/2$  instead of  $V_{tn}$  before the other storage node can be affected. A more stable memory cell is harder to write and requires additional area. The increased p-channel transistor size increases the die size by approximately 7%.

In UTMIC's 64K SRAM, the write time specification can be met even with the more stable memory cell because the write circuitry forces the memory cell columns to  $V_{dd}$  and Gnd during a write operation. Although the write circuitry that provides  $V_{dd}$  and Gnd is larger than a standard write circuit that only provides Gnd, it increases the die size by less than 0.5%.

The 64K SRAM uses high-valued polysilicon resistors in series with the gates of both cross-coupled inverters and increased capacitance on the storage nodes to increase the

feedback switching time. Increased stability and increased feedback switching time provide improved SEP protection. The 64K SRAM can use a lower valued resistor because the memory cell is more stable than a typical SRAM. The lower valued resistor is more manufacturable and is less affected by temperature. The polysilicon resistor process provides a tighter than typical control over the resistor value. This tight resistance control provides greater SEP protection over a wider temperature range and also allows shorter write times. Including a resistor in the memory cell increases the die size by less than 9%. The cross-coupled resistors are incorporated in the single layer of polysilicon which also forms the gates of the transistors. Metal contacts for power and ground are incorporated in every cell to help collect some of the charge deposited by a particle passing through a nearby junction.

The effect of the cross-coupled resistors is to increase the threshold LET for SEP upset and reduce the effective saturated cross-section of the device. [9] These effects for UTMIC's 64K SRAM are shown in figure 5 and 6. Using the LET threshold and effective cross-section from these graphs, the error rate in errors/bit/day as a function of resistor value may be calculated for any space environment using CREME or SpaceRad as described above. The error rate at 125 C (worst case) for UTMIC's 64K SRAM is shown in figure 7 as a function of resistor value. By screening devices at the wafer level for resistor value and p-channel drive current, we can guarantee an error rate of less than  $1.0E-10$  errors per bit per day at 125 C.

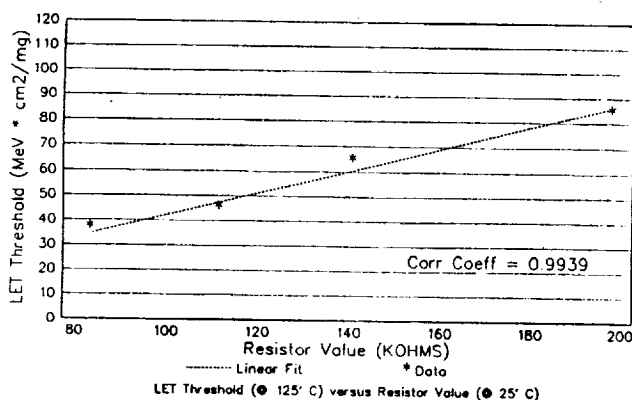


Figure 5: LET Threshold vs. Resistor Value

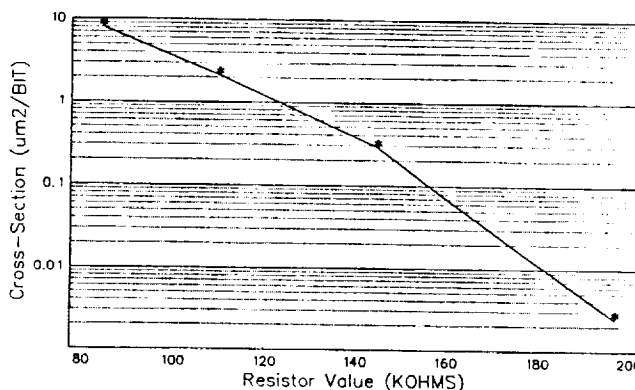


Figure 6: Error Rate vs. Resistor Value

## 6.2 SEP-Hardened Flip-Flop Design

In logic systems, storage nodes such as flip-flops must retain data reliably, or the integrity of the logic system can be severely compromised. A SEP-induced upset of a single bit in a microprocessor register can send the system into an irrecoverable state. Detection of these types of errors can require substantial overhead in software and hardware complexity and

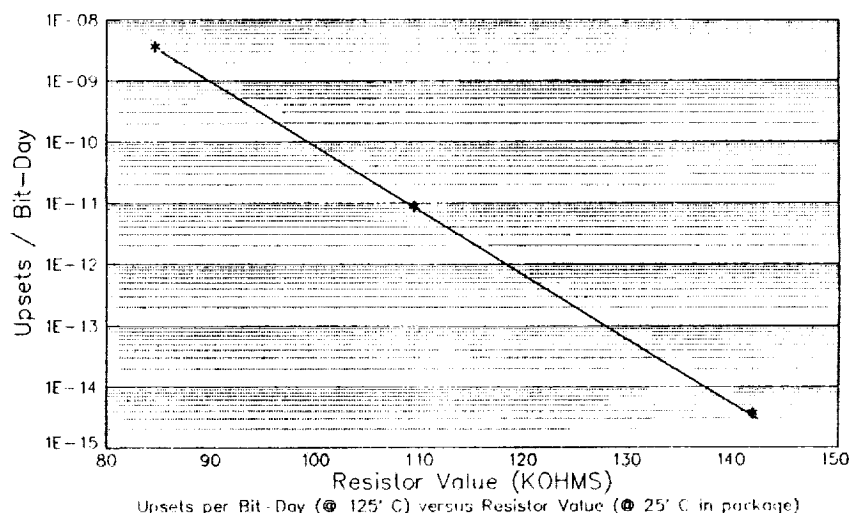


Figure 7: Error Rate vs. Resistor Value

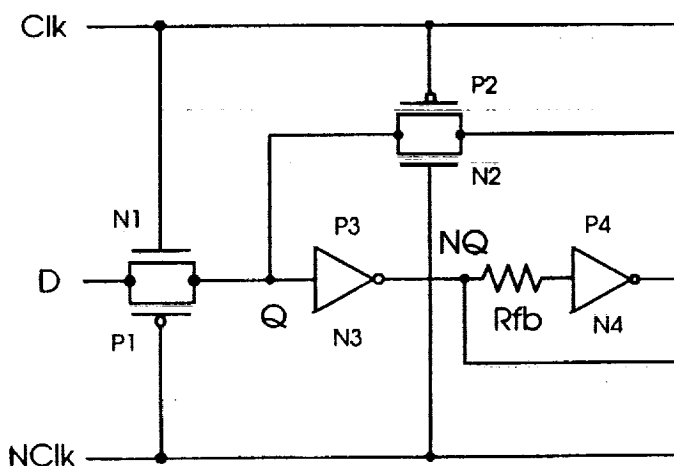


Figure 8: Resistor Application in Flip-flop

is possible to substantially improve the SEP performance without introducing additional processing complexity. However, there is usually a penalty in increased die area.

If the wafer fabrication process technology has poly-resistors available (used for SEP-hardening SRAM cells described above), these resistors may be used to increase flip-flop hardness as shown in figure 8. This technique can result in some performance degradation of the flip-flop over temperature as reported by Sexton et al. [10] If resistors are not available, circuit techniques, coupled with the simulation techniques similar to those described above, can be used to develop flip-flop register cells which have improved SEP performance over conventional flip-flop circuits. [11,12]

To determine the effectiveness of the simulation techniques described above, we simulated the SEP upset for a number of the flip-flop cells in UTM's gate-array library. The simulations for one of these cells, DFAPCB - a D-type flip-flop whose logic diagram is shown in figure 9, was compared with experimental data from heavy ion tests performed at Brookhaven National Laboratory. To accurately determine the upset rate of the flip-flop,



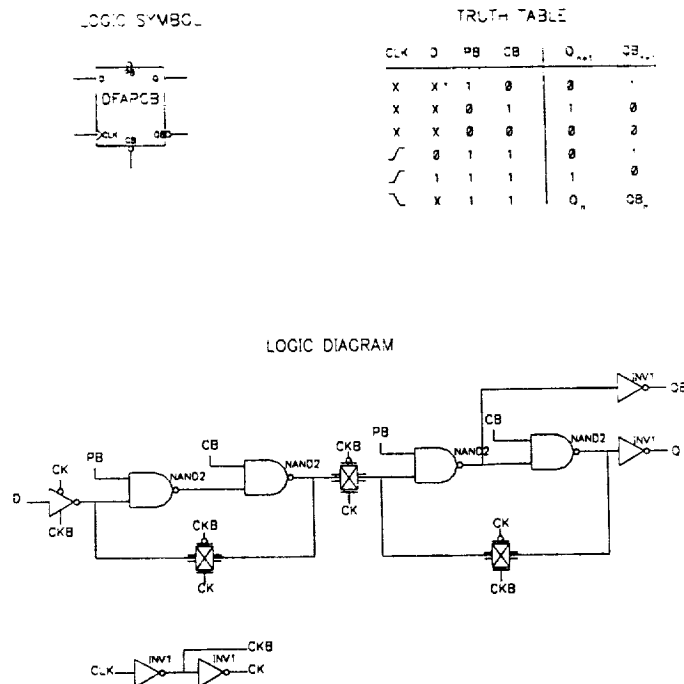


Figure 9: D-type Flip-flop Logic Diagram

it was necessary to determine the effective critical charge (the charge required to upset the state of the flip-flop) for every node within the circuit for both a high-node state and a low-node state. The effective critical charge, along with the junction area for each node, was then used to determine the upset rate for the node using the SpaceRad program. The sum of the upset rates for all nodes was then taken as the upset rate for the flip-flop. The simulated error rate for the DFAPCB flip-flop was  $4.24\text{E-}8$  errors per cell per day. This compares to an experimentally determined error rate of  $3.38\text{E-}8$  errors per cell per day for this flip-flop cell.

We have also developed SEP-upset improved cells which support our radiation-hard gate array cell library. Some of the cells are capable of providing an error rate of  $1.0\text{E-}10$  errors per cell per day. Fully redundant cells have also been designed which require twice as many transistors as a non-redundant design, but are SEP-immune. The results of this work have provided a number of guidelines for selecting and designing an SEP-hard flip-flop. These guidelines are discussed in Table 2.

## 7 Conclusions

Use of the simulation techniques described in this paper substantially increases the confidence that a design will meet its objectives for SEP hardness, and the cell layout can be optimized without compromising circuit performance. We have demonstrated an SEP error rate less than  $1.0\text{E-}10$  errors/bit-day for a 90% worst-case geosynchronous orbit environment over the entire  $-55\text{ C}$  to  $+125\text{ C}$  temperature range for a rad-hard 64K SRAM

Problem	Flip-Flop Design Approach
Minimize stacked p-channel devices	Remove NOR gates and replace with NAND gates.
Eliminate transmission gates	Remove transmission gates and replace with clocked inverters.
Minimize sensitive node area	Simplify design as much as possible consistent with functional requirements. Add redundant (parallel) transistors on sensitive nodes internal to the cell.

Table 2: Design Considerations for Improving SEP Hardness of Flip-Flops

while maintaining a less than 55 ns cycle time. We have also demonstrated the capability to model the SEP error rate of gate array cells and have applied the simulation and design techniques described in this paper to develop SEP-tolerant and SEP-hard flip-flop designs.

SEP hardness of integrated circuits cannot be assured by screening commercial devices, or by normal system-level or logic-level design techniques. Good SEP hardness can only be obtained by using a wafer fabrication process which provides the proper characteristics and proper attention to good design practices at the transistor level. Since commercial semiconductor manufacturers do not consider SEP effects when designing their circuits, it will be necessary to develop custom circuits which are designed for SEP hardness for mission-critical applications.

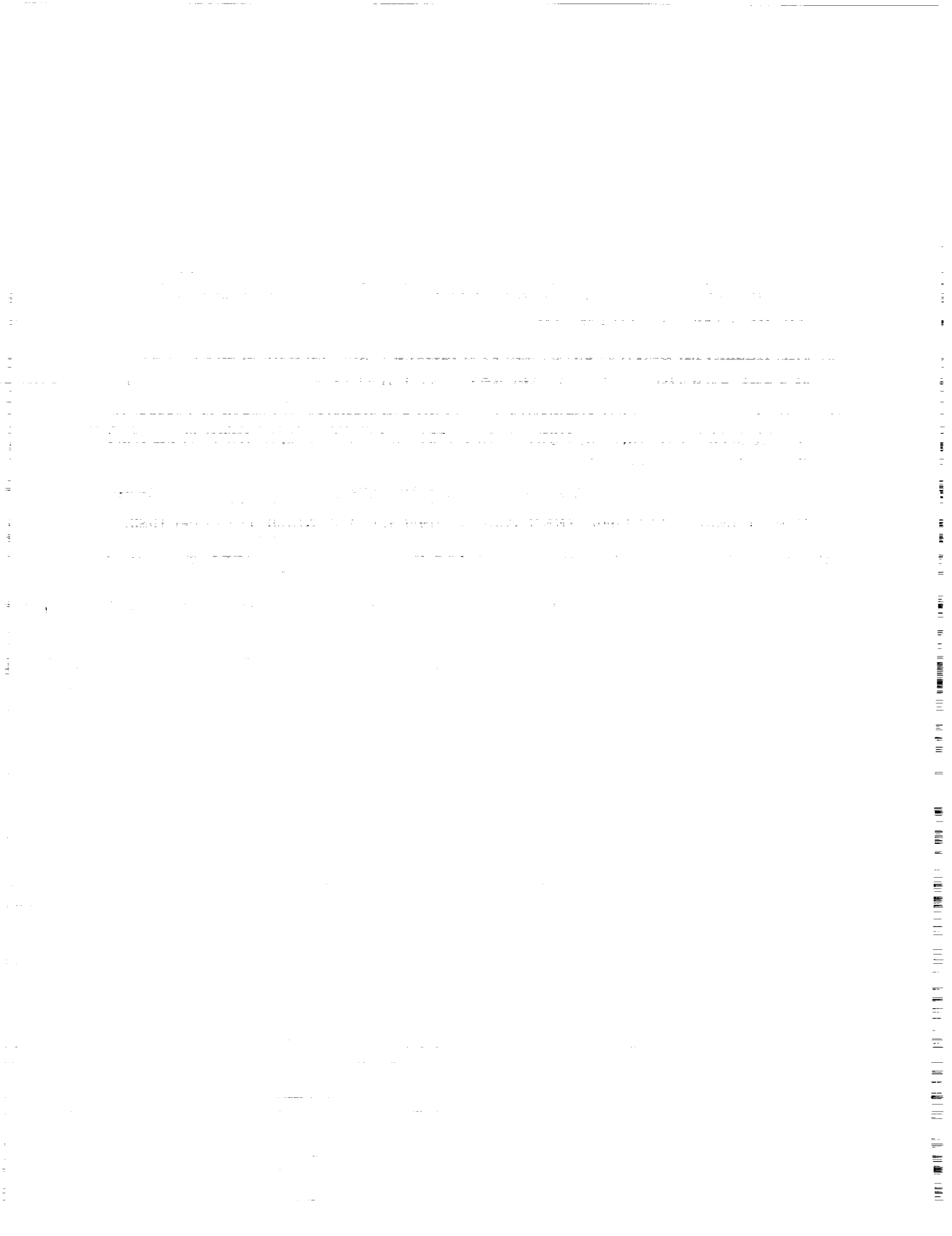
## 8 Acknowledgments

The authors would like to thank John Silver for providing valuable inputs to this paper.

## References

- [1] TMA DAVINCI is a trademark of Technology Modeling Associates, Inc.
- [2] J. J. Silver, Circuit Designs for Reliable Operation in Hazardous Environments, *NSREC Tutorial Short Course*, July 1987 and references therein.
- [3] A. Knudson, A. Campbell, Comparison of Experimental Charge Collection Waveforms with PISCES Calculations, *IEEE Trans Nucl Sci.*, NS-38, 1991.
- [4] J. Rollins et al., Cost-Effective Numerical Simulation of SEU, *IEEE Trans Nucl Sci.*, NS-35, 1988.
- [5] J. H. Adams Jr., R. Silberberg, and C. H. Tsao, Cosmic Ray Effects on Microelectronics, *NRL Memorandum Report H506*, Naval Research Laboratory, Washington, D.C.

- [6] SpaceRad is a product of Space Radiation, Severn Communications Corp.
- [7] S. E. Diehl, et al., Error Analysis and Prevention of Cosmic Ion-Induced Soft Errors in Static CMOS RAMs, *IEEE Trans on Nucl Sci*, Vol. NS-29, December 1982, page 2032.
- [8] L. Alles, K.L. Jones, J.E. Clark, J.C. Lee, W.F. Kraus, S.E. Kerns, L.W. Massengil, Rad-Hard SOI/SRAM Design Using a Predictive SEU Device Model, *Digest of Papers, GOMAC*, Vol. 16, 1990, page 443.
- [9] J.R. Adams, M. Barry, J. Silver, and P. Rudeck, Design, Simulation, and Evaluation of a SEP-hard SRAM Memory Cell, *RADECS*, August 1991.
- [10] F.W. Sexton, et al., SEU Simulation and Testing of Resistor-Hardened D-Latches in the SA-3300 Microprocessor NSREC, July 1991, to be published in *IEEE Trans Nucl Sci.*, 1991, and references therein.
- [11] S. E. Diehl, J. E. Vinson, B.D. Shafer, and T. M. Mnich, Considerations for Single Event Immune VLSI Logic, *IEEE Trans on Nucl Sci.*, Vol. NS-30, December 1983.
- [12] L.R. Rockett, Jr., An SEU-Hardened CMOS Data Latch Design, *IEEE Trans Nucl Sci.*, Vol. 35, No. 6, December, 1988.



## Pulse-Firing Winner-Take-All Networks

Jack L. Meador

School of Electrical Engineering and Computer Science  
Washington State University  
Pullman WA, 99164-2752

**Abstract-** Winner-take-all (WTA) neural networks using pulse-firing processing elements are introduced. In the pulse-firing WTA (PWTA) networks described, input and activation signal shunting is controlled by one shared lateral inhibition signal. This organization yields an  $O(n)$  area complexity that is convenient for integrated circuit implementation. Appropriately specified network parameters allow for the accurate continuous evaluation of inputs using a signal representation compatible with established pulse-firing neural network implementations.

### 1 Introduction

The winner-take-all (WTA) function plays a central role in competitive neural networks and is related to recurrent on-center off-surround models of natural neural systems [1-3]. Although it can be realized sequentially via pairwise comparisons, the WTA operation is more effectively realized in parallel analog circuits via a distributed network of processing elements which compare relative input magnitudes and allow only that element with the largest input (or "winner") to remain active. Parallel analog WTA realizations have been described which use Hopfield Network dynamics [4], and MOS current conveyors [5,6]. The model introduced in this paper and its electronic implementation are more like a WTA mechanism inspired by natural presynaptic inhibition feedback [7]. The new pulse-firing WTA (PWTA) model employs a unique combination of a self-shunting feedback term with output hysteresis to yield a WTA network compatible with asynchronous pulse-firing neural network implementations described variously as impulse, pulse-stream, and neural-type networks [8-10].

This paper first introduces asynchronous pulse firing processing units in Section 2. These are the basic computational units used in PWTA networks. The mathematical foundation of PWTA networks is then presented in Section 3 where the system dynamics of a general PWTA network are developed. Section 4 continues with the presentation of MOS circuit implementations. Section 5 closes with an analysis of finite circuit precision effects.

### 2 Asynchronous Pulse Firing Processing Units

The dynamics of the pulse firing processing units used in a PWTA network obey the following equations:

$$\begin{aligned}\kappa \dot{v} &= -\alpha v + x - (\beta v + x - \gamma)g(v), & v(t_0) &= 0 \\ y &= g(v)\end{aligned}\tag{1}$$

where  $v$  is unit activation,  $x$  is total unit input,  $y$  is unit output, and  $g(\cdot)$  is the binary hysteresis function shown in Figure 1. As can be deduced from the figure  $g(v)$  includes as a special case the simple threshold nonlinearity (when  $V_{tl} = V_{th}$ ) although the specific pulse firing dynamics described here would cease to exist in that situation. Throughout this paper input  $x$  is assumed to be positive and time variant. The unit response to a constant input is a train of regularly spaced constant width pulses. The larger the input signal  $x$ , the greater the output pulse repetition rate. The parameter  $\alpha$  establishes a first-order response to  $x$  during the input integration phase of operation as defined by the absence of an output pulse ( $y = 0$ ). That response is shifted to one defined by  $\alpha + \beta$  during the firing phase, as defined by the presence of an output pulse ( $y = 1$ ). Since  $x$  is shunted during the output pulse period, processing element state asymptotically approaches  $\epsilon = \gamma/(\alpha + \beta)$ . Parameter  $\kappa$  uniformly scales all unit time constants. One pulse firing cycle is summarized by the integration of the input signal until  $v$  reaches  $V_{th}$ , whereupon the switches toggle, causing the discharge of  $v$  to  $\min(\epsilon, V_{tl})$ . Oscillation is sustained provided  $\epsilon < V_{tl}$ .

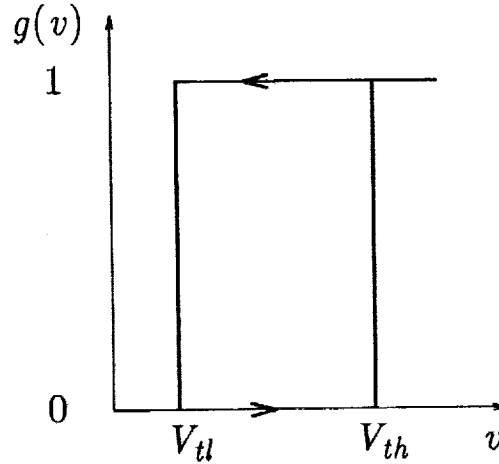


Figure 1: Output hysteresis function

### 3 PwTA Network Dynamics

A PwTA network combines the unit dynamics described in (2) with lateral inhibition. Lateral inhibition from a combination of unit outputs can be expressed in a form which yields network state equations similar to those of the presynaptic inhibition model described by Yuille and Grzywacz [7]:

$$\kappa \dot{v}_i = -\alpha v_i + x_i F(\mathbf{V}) - H(\mathbf{V})\tag{2}$$

where

$$F(\mathbf{V}) = 1 - \bigvee_k g(v_k)$$

and

$$H(\mathbf{V}) = ((\lambda - \beta)v_i + \gamma - \varsigma)g(v_i) + (\beta v_i - \gamma) \bigvee_k g(v_k)$$

with  $\bigvee$  indicating the logical OR operation and  $\mathbf{V}$  corresponding to the vector of unit activations.  $F(\mathbf{V})$  is a binary value establishing the input inhibition which occurs while any processing element generates a pulse. It is during this "output firing phase" of the network that  $H(\mathbf{V})$  controls the processing elements such that a winning unit activation decays at a different rate to a different equilibrium than that of a losing unit. The model parameters allow for the independent adjustment of winning and losing unit decay rates and asymptotes. Properly chosen parameter values guarantee WTA function independent of initial system state without the need for external synchronization [11].

All units contribute to a shared lateral inhibition signal identically. The winning output is indicated by the dominance of the first unit activation to reach  $V_{th}$ : since it establishes the synchronized re-initialization of all units, it is the only one to fire. In general, the winning unit is determined by a combination of initial network activation state and input magnitude. With appropriately chosen parameters, however the reset state establishes initial conditions which make the winning unit decision independent of initial state and dependent exclusively upon the  $x_i$  inputs.

For the winning unit to exactly correspond to the one having the largest input, it is important that initial condition independence be maintained. To guarantee this independence in the PWTA network described by (2) parameters are chosen such that all units reset to an identical initial condition.

All activations in the PWTA of (2) will reset to near-identical initial states if parameters are chosen such that  $v_i^* < V_{th}$ ,  $v_k^* = V_{th}$  and  $\beta \ll \lambda$  [11]. During the output firing phase, these values cause the losing units to approach  $V_{th}$  well before the winning unit does. When the winning unit reaches  $V_{th}$ , it terminates the output firing phase and all activations cease to decay. Theoretically, the losing units only asymptotically converge to  $V_{th}$  while the winning unit converges via a truncated exponential. Even though this is mathematically imprecise, in a practical sense it can be assumed that  $\beta$  is chosen large enough with respect to  $\lambda$  for losing units to converge to within the limits of finite precision hardware well before the firing phase ends.

A geometric interpretation of ideal PWTA network operation with constant inputs is illustrated in Figure 2. Each loop in the state diagram corresponds to one firing cycle. Unit activations are reset to  $V_{th}$  at state  $S_0$  in the diagram. The input integration phase (1 and 3 in the figure) begins at  $S_0$  and terminates when the  $V_{th}$  threshold is reached. That is followed by the output firing phase (2 and 4 in the figure) during which unit activations

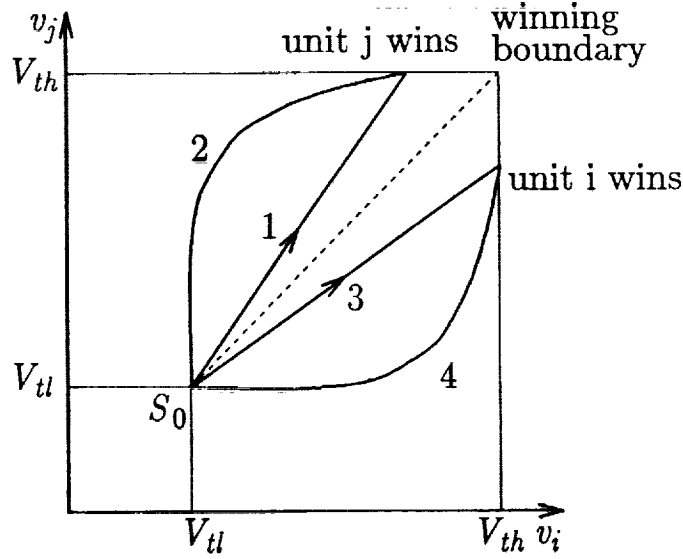


Figure 2: Ideal PWTa network operation

decay. In the figure, trajectory 1-2 corresponds to the path followed when unit  $j$  wins, and trajectory 3-4 to when unit  $i$  wins. Which unit wins is determined by the one which first reaches  $V_{th}$ . That in turn is determined by the state trajectory during the input integration phase. With constant inputs, that trajectory is linear with slope proportional to the quotient of the input signal magnitudes. A “winning boundary” for constant inputs can be identified by unit slope (dashed line in the figure). This geometric interpretation of PWTa operation will prove useful in a later discussion of finite precision effects.

Figure 3 illustrates the operation of a 2-unit PWTa network in response to a smooth transition between two inputs. In this example, input signals  $x_1$  and  $x_2$  move from 0 to 1 and from 1 to 0 respectively, crossing at  $t=10$ . The parameters chosen for this simulation are  $V_{tl} = 1$ ,  $V_{th} = 4$ ,  $\kappa = 0.1$ ,  $\alpha = 0.1$ ,  $\beta = 1.2$ ,  $\gamma = 1.3$ , and  $\lambda = 0.6$ . The activation state space diagram for this simulation is shown in Figure 4. It can be seen how the reset state with these parameters assures input order preservation.

## 4 CMOS Circuit Implementation

By way of introduction to the CMOS PWTa network, a CMOS implementation of a pulse firing processing element shall first be considered. Figure 5 shows the circuit for an impulse neural circuit as described previously in [8]. For simplicity,  $\gamma = 0$ , and  $\alpha$  is for practical purposes nonexistent by virtue of the low leakage currents exhibited in MOS technology.  $C_\kappa$  includes not only the ideal capacitance of a poly-1 capacitor, but also stray wiring capacitance and the input capacitance exhibited by the Schmitt trigger  $G$ . The Schmitt trigger provides high voltage gain at the threshold voltages  $V_{tl}$  and  $V_{th}$ , with positive feedback from the output establishing the active threshold. The Schmitt trigger output can be expressed in terms of the hysteresis function  $g$  of (2) as  $G(v) = V_{DD}g(v)$ .  $\beta$  corresponds to the channel conductance of M2 which operates in the active region when an output pulse is generated. Further details regarding the operation of this circuit are



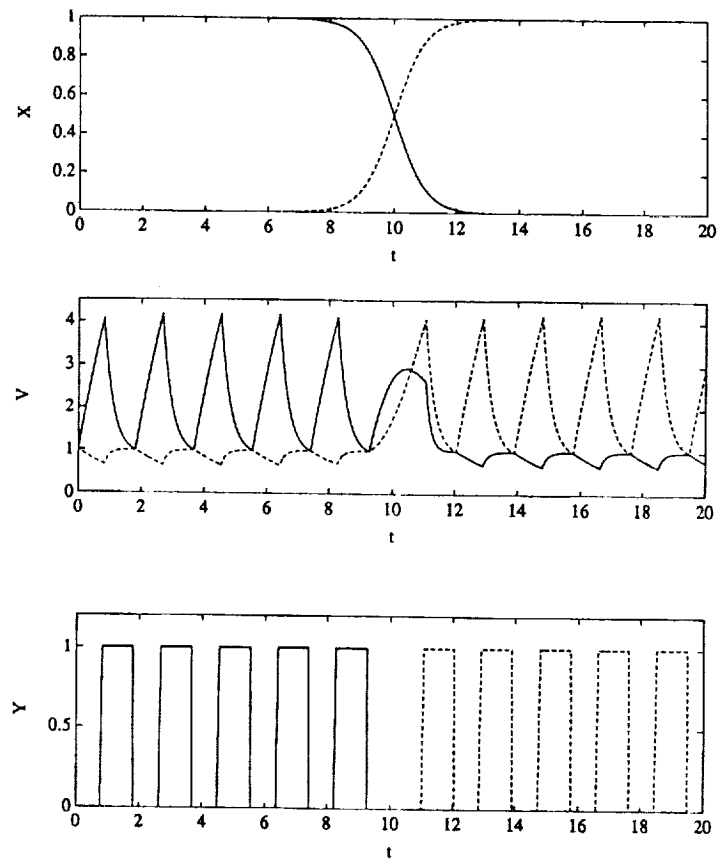


Figure 3: Inputs, state variables and output of a 2-unit PWTa network

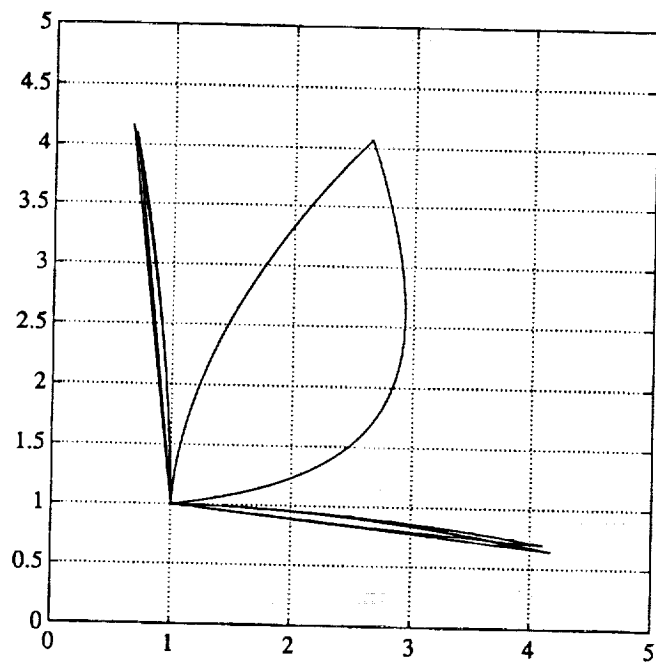


Figure 4: State trajectory of a 2-unit PWTa network simulation

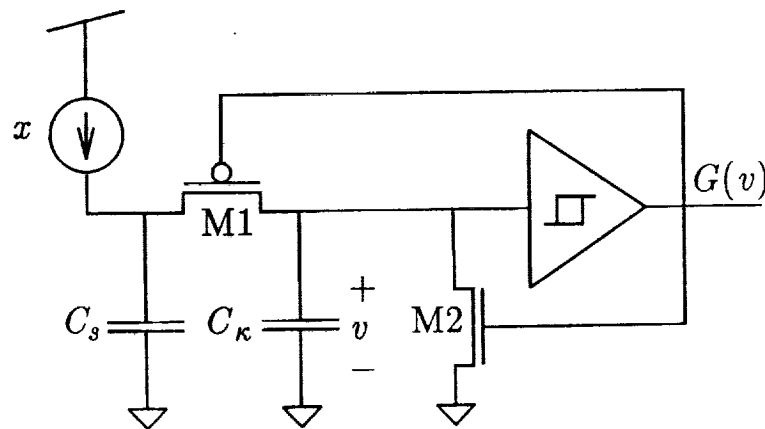


Figure 5: CMOS implementation of a pulse firing processing unit

provided in [8].

A CMOS implementation of a PWTA cell is shown in Figure 6. The basic elements of the CMOS impulse circuit are augmented by additional MOSFETs which establish various parameters associated with the ideal model. Variations of this circuit having reduced transistor counts are also possible [11]. The circuit of Figure 6 simply represents the most general CMOS PWTA implementation consistent with the Equation (3) definition.

Two local signals and one global signal control circuit operation in accordance with current network state. The local signals  $G_i$  and  $\overline{G}_i$  indicate that unit  $i$  is the winning unit when  $G_i = V_{DD}$  and  $\overline{G}_i = 0V$ . These signals select the local firing response. Both the true and complemented global lateral inhibition signal  $F$  and  $\overline{F}$  are derived by a pseudo-NMOS NOR gate and a CMOS inverter consisting of transistors M11 through M14 in the diagram. These signals are distributed on two wires between all cells of the WTA network. NOR pulldown transistors (M11) are distributed across all cells while there need only exist a single pullup transistor (M12) and single inverter (M13, M14). When any unit in the network initiates a pulse,  $F$  becomes active, causing all units to enter the output firing phase.

Transistor M1 disconnects input current  $x_i$  during the output firing phase, causing it to be shunted into the parallel capacitance of some input circuit (not shown – see [8] for further details). Also during the firing phase, transistors M2, M7, M6, and M10 conduct, allowing some combination of the currents  $I_1$  through  $I_4$  to flow.

The circuit branch consisting of M2 through M4 establishes a current which corresponds to the constant  $I_1 = \gamma$ . Similarly, branch M7 through M9 establishes a constant current corresponding to  $I_3 = \zeta$ . Ignoring the nonlinear component of channel conductance, the branch consisting of transistor M10 establishes a current corresponding to  $I_4 = \lambda v_i$  and the M5, M6 branch a current analogous to  $(\beta - \lambda)v_i$ . As with the circuit of Figure 5,  $\alpha$  is assumed to be negligible.

During the output phase, the signals  $G$  and  $\overline{G}$  control the unit response. If the unit is a winner, then  $G = V_{DD}$ ,  $\overline{G} = 0V$ , and branch currents  $I_3$  and  $I_4$  are allowed

to flow. This establishes a winning unit response where the unit activation asymptotically decays to  $\zeta/\lambda$ , but is truncated at  $V_u$  when the winning unit terminates the output firing phase. If the unit is a loser, then  $G = 0V$ ,  $\bar{G} = V_{DD}$ , and branch currents  $I_1$ ,  $I_2$ , and  $I_3$  are allowed to flow. This establishes a losing unit response, where the unit activation asymptotically approaches  $V_u$  at a much faster rate than it would otherwise as the winning unit.

## 5 Finite Precision Effects

Thus far, the effects of finite parameter precision have been ignored. Intra-cell parameter variation will contribute to deviations from the ideal performance previously described. This section focuses upon the parametric variations which will have the greatest effect upon PWTAs performance that also are the most likely to occur in contemporary CMOS fabrication processes.

The overall function of a PWTAs network is to select the input signal having the greatest magnitude. Inspection of Figure 2 reveals that there are two potential error sources which can interfere with that function. These are errors in the determination of the initial network state,  $S_0$  and deviations in the position of the winning boundary. These variations effectively give an unfair advantage to some processing units, sometimes allowing units to fire even though their inputs are not necessarily the largest. Fortunately, it can be shown that this occurs only when two inputs have very nearly the same magnitude. Units having input signals which are "clearly" not the largest will remain quiescent. The definition of "clearly" is expressed as a hysteresis deadband which naturally occurs around the winning boundary. This hysteresis arises directly from parametric variation.

For the remainder of this section only constant inputs will be considered. This allows for the analysis of parameter precision effects while the network is in a steady-state operating condition. Figure 2 illustrates network operation under ideal conditions when the critical parameters  $V_u$ ,  $V_{th}$ ,  $\kappa$ ,  $\gamma$  and  $\beta$  are assumed identical across all units. Under these conditions, unit  $i$  wins if

$$\frac{dv_i}{dv_j} > 1 \quad (14)$$

with unit  $j$  winning otherwise.  $\frac{dv_i}{dv_j} = 1$  corresponds to the ideal winning boundary (dashed line) of Figure 2.

Variations in the scaling constant  $\kappa$  yield an inaccurate winning boundary definition.  $\kappa$  is determined in the CMOS implementation by the MOS capacitor  $C_\kappa$  in the previous circuit diagram. Variations in capacitor geometry will lead to inter-unit  $\kappa$  variation and subsequently give those units having a smaller  $\kappa$  an advantage in the race toward  $V_{th}$ . Recognizing that

$$\frac{dv_i}{dv_j} = \frac{\kappa_i \dot{v}_i}{\kappa_j \dot{v}_j} \quad (15)$$

yields the decision rule that unit  $i$  wins if

$$\frac{dv_i}{dv_j} > \frac{\kappa_j}{\kappa_i} \quad (16)$$

This rule reduces to the ideal one of (14) when  $\kappa_i = \kappa_j$ . Although all units are initialized to the same state at  $S_0$ , the decision boundary is shifted such that one unit is favored over another.

Variation of  $C_\kappa$  alone leads to finite precision for the winning boundary. Parameter variations which affect the initial network state and the unit firing thresholds lead to more complex hysteresis effects at the winning boundary. Firing thresholds, as defined by  $V_{th}$  in the Schmitt trigger are typically determined by device geometries. The initial network state is determined in part by  $V_{ti}$  which is also dependent upon Schmitt device geometries. The other part of initial network state is determined by the geometries of M2-M6 and M10 of Figure 6 (corresponding to  $\beta$ ,  $\gamma$  and  $\lambda$  in the ideal equations). Not only do such variations change the slope of the winning boundary, but the slope is also dependent upon the winning unit as well. Under these conditions, the current winner is favored by the initial states such that a new winner must have a significantly larger input than the present one. These effects can be used to extend the decision rule expressed by (16) to one where unit  $i$  wins if

$$\frac{dv_i}{dv_j} > \frac{\kappa_j}{\kappa_i} \frac{V_{thi} - V_{ti}}{V_{thj} - V_{0j}} \quad (17)$$

where  $V_{0i}$  and  $V_{0j}$  are determined by the combined variations of M2-M6 and M10 between units  $i$  and  $j$ . It can be easily verified that this decision rule reduces to the ideal case when there is no inter-unit variation. The effect this has on overall PWTa function is to introduce a hysteresis deadband which only affects close decisions.

## 6 Conclusion

An ideal linear model has been used to establish a general basis for PWTa function. This model improves an earlier one based upon presynaptic inhibition in two ways. The new model uses  $O(n)$  interconnect for lateral inhibition and does not require an external reset signal since it is fully asynchronous. Furthermore, it provides information regarding how strong a winning input is – a feature not always found in winner-take-all networks. Model parameters can be chosen to guarantee ideal winner-take-all function given precise parameter specifications. The model is also fully compatible with previously established asynchronous pulse firing analog neural ICs.

CMOS PWTa circuits have also been presented. These circuits necessarily deviate from the ideal linear model, but they can be designed to exhibit similar behavior simply by accounting for the nonlinear characteristics of the electronic devices they employ. Non-ideal effects arising from practical implementation considerations have also been addressed.

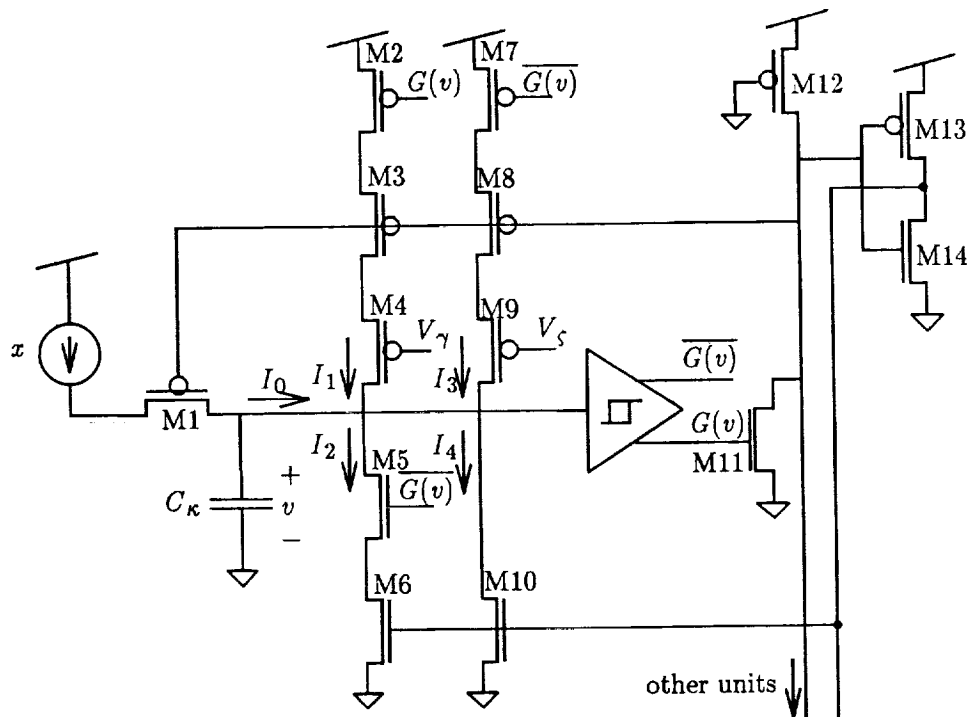


Figure 6: A genral CMOS PWTa cell

## References

- [1] J. Hertz, A. Krogh, and R.G. Palmer, Introduction to the Theory of Neural Computation, 217-228, Addison Wesley, 1991.
- [2] S. Grossberg, Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks, *Studies in Applied Mathematics*, Vol. 12, 213-257, MIT Press, 1973.
- [3] S. Grossberg, Nonlinear Neural Networks: Principles, Mechanisms, and Architectures, *Neural Networks*, Vol. 1, 17-61, Pergamon Press, 1988.
- [4] E. Majani, R. Erlanson, and Y. Abu-Mostafa, On The K-Winners-Take-All Network, *Progress in Neural Information Processing Systems*, Vol. 1, 635-642, Morgan Kaufman, 1988.
- [5] J. Lazzaro, et al., Winner-Take-All Networks of  $O(n)$  Complexity, *Progress in Neural Information Processing Systems*, Vol. 1, 703-711, Morgan Kaufman, 1988.

- [6] A. Andreou, et al., Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems, *IEEE Trans. on Neural Networks*, Vol. 2, 205-213, IEEE Press, 1991.
- [7] A. L. Yuille, and N. Grzywacz, A Winner-Take-All Mechanism Based on Presynaptic Inhibition Feedback, *Neural Computation*, Vol. 1, 335-347, MIT Press, 1989.
- [8] J. Meador, et al., Programmable Impulse Neural Circuits, *IEEE Trans. on Neural Networks*, Vol. 2, 101-109, IEEE Press, 1991.
- [9] A. F. Murray, et al., Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques, *IEEE Trans. on Neural Networks*, Vol. 2, 193-204, IEEE Press, 1991.
- [10] G. Moon, et al., Analysis and Operation of a Neural-Type Cell (NTC), *Proc. IEEE ISCAS*, pp. 2332-2334, IEEE Press, 1991.
- [11] J. Meador, Pulse Firing Winner-Take-All Networks, *Internal Manuscript, School of Electrical Engineering and Computer Science, Washington State University, Pullman WA.*, June, 1991.

## Training Product Unit Neural Networks with Genetic Algorithms

D. J. Janson and J. F. Frenzel

Department of Electrical Engineering

email: jff@k2.ee.uidaho.edu

email: djanson@bashful.ee.uidaho.edu

and D. C. Thelen

Microelectronics Research Center

email: dthelen@granite.mrc.uidaho.edu

University of Idaho, Moscow, Idaho 83843

**Abstract-** This paper discusses the training of product neural networks using genetic algorithms. Two unusual neural network techniques are combined; product units are employed instead of the traditional summing units and genetic algorithms train the network rather than backpropagation. As an example, a neural network is trained to calculate the optimum width of transistors in a CMOS switch. It is shown how local minima affect the performance of a genetic algorithm, and one method of overcoming this is presented.

### 1 Introduction

Neural networks have been applied successfully to many problems in recent years. Traditionally these networks are composed of multiple layers of summation units. These simple units sum their inputs, each input multiplied by a variable weight. This summation is usually then squashed by a non-linear equation such as the logistic function. Several researchers have shown that networks composed of these units can calculate any function to any arbitrary degree of accuracy given enough summation units. [1] However, there are many functions that are complicated enough that the number of summation units it takes to duplicate them are prohibitive. One very commonly found task is that of higher order combinations of the inputs such as either  $X * X$  or  $X * Y$ .

One proposed solution is a new unit called the "sigma-pi unit" [3]. This unit not only applies a weight to each input, but also applies a weight to the second and possibly higher order products of the inputs. While much more powerful than the traditional summation unit, the number of weights increase very rapidly with the number of inputs, and soon become unmanageable when applied to solving large problems. Since most problems only need one, or at most a few, of these terms, the sigma-pi unit is overkill.

#### 1.1 Product Units

A suitable alternative was introduced by Durbin and Rumelhart [2]. The "product unit" computes the product of its inputs, each raised to a variable power. This is shown in

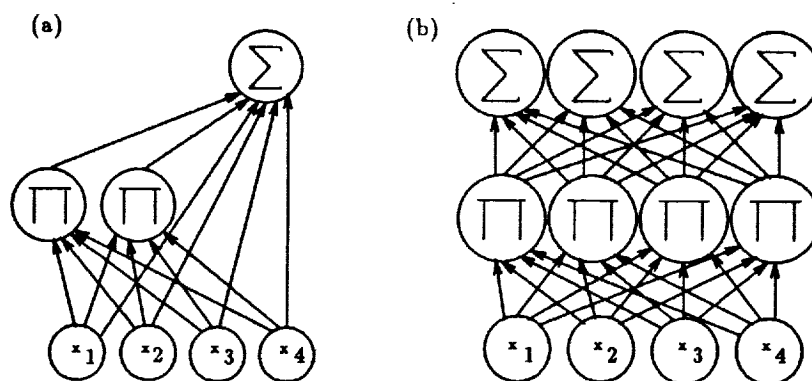


Figure 1: Recommended product network configurations [4]

Equation 1.

$$y = \prod_{i=1}^N X(i)^{p(i)} \quad (1)$$

The  $p(i)$  term is treated in the same way as the variable weights for summation units. Using the modified version of backpropagation presented by Durbin and Rumelhart, these product units can provide much more generality than sigma-pi units. While a sigma-pi unit is constrained to using just polynomial terms, the product units can use fractional and even negative terms. As Durbin and Rumelhart point out, product units can actually be considered a superset of sigma-pi units; for if several of the product units are used, and they are constrained to only integer values, they would have the same results.

There are many ways that product units can be used in a network. However, the overhead required to raise an arbitrary base to an arbitrary power makes it unlikely that they will replace summation units. Durbin and Rumelhart propose that the primary use of the product units will be to supplement the power of the summation units. Two proposed architectures are shown in Figure 1. The term product neural networks (or product networks) will be used to refer to networks containing both product and summation units.

While product units increase the capability of a neural network, they also add complications. Not only is backpropagation harder to accomplish, but the solution space becomes more convoluted. As Durbin and Rumelhart pointed out, there are often local minima that trap the network. As a possible solution to this problem, this paper investigates the use of genetic algorithms to train product networks.

## 2 Genetic Algorithms

### 2.1 Introduction

A genetic algorithm (GA) is an exploratory procedure that is able to locate near-optimal solutions to complex problems. To do this, it maintains a set (called a population) of trial solutions (called chromosomes). Through a repeated four-step process, these chromosomes



evolve until an acceptable solution is found. These steps are evaluation, reproduction, breeding, and mutation. A representation for possible solutions must first be developed. Then, with an initial random population, the GA is able to solve the problem almost without regard to the interpretation of the chromosome. Each generation, the chromosomes produced, through survival-of-the-fittest and exploitation of old knowledge in the gene pool, should have an improved ability to solve the problem.

There were two primary reasons why GAs were applied to training product networks. First was that the addition of product nodes made the solution space more complicated. Because backpropagation is a gradient decent algorithm, it is very likely to get caught in a local minimum. The second reason was that backpropagation tends to be slow with complicated problems. It often takes many iterations to train a complicated network. It is hoped that the use of a GA will find the best answer much faster than backpropagation.

## 2.2 Representation

Before applying a genetic algorithm to any task, a representation for possible solutions must be found. The most common method for representing these possible solutions is with a bit string. Higher order strings (such as character strings) or trees (such as binary trees) have also been used. Since the architecture of the product networks to be trained will be known, a binary string representation with a fixed number of bits per weight can be constructed. Thus, each weight in the network has a certain number of bits associated with it. This representation permits each chromosome to be decoded easily, while still allowing each weight a large degree of freedom. The typical generation used had between 30 to 100 members in a population, with 16 bits representing a weight.

## 2.3 Evaluation

The first step in any generation is the evaluation of the current chromosomes. This is the only step where the interpretation of the chromosome is used. Each chromosome in the population is decoded, and the result is used to solve the original problem. This solution is then graded on how well it solved the problem. The method used to grade product networks is to calculate the sum of squared error (SSE) for the training set. The fitness of the chromosome is equal to  $1/(1 + \text{SSE})$ . This means that the better a network performs, the higher its fitness, with a perfect network having a fitness of 1.

## 2.4 Reproduction

The next step in a generation is to create a new population based upon the evaluation of the previous one. Every chromosome generates a specific number of copies of itself, based on how well it solved the problem. Thus the chromosomes that performed better will produce several copies of themselves, while the worst chromosomes won't produce any copies. This is the step that allows GAs to take advantage of a survival-of-the-fittest strategy.

There are several methods to calculate the number of offspring that each chromosome will have. One of the more prevalent methods is called ratioing. With ratioing, each chromosome produces a number of offspring directly related to its fitness, with the only restriction being that the total number of chromosomes per generation remains constant. Thus, if one chromosome has a fitness that is twice that of another, then the superior chromosome would produce twice as many offspring. However, there are two major problems with this method. First, if all the chromosomes have similar fitness, each member in the population will produce one offspring. This results in little pressure toward improving the solution. The second problem, although from a different source, has the same effect. If any one chromosome should happen to have a fitness much larger than any of the others, then that chromosome would create most, if not all of the new offspring. This discriminates against the remaining information of the gene pool in favor of this super-chromosome, losing the information in the gene-pool. This particular type of stagnation has been labeled premature convergence.

The method the author used to train the product networks is ranking [5]. In ranking, the whole population is sorted by fitness. The number of offspring each chromosome will generate is then determined by where it falls in the population. The ranking algorithm used was that the top 30% of the population generated two offspring each, the bottom 30% of the population generated no offspring, and the rest of the population each generated one offspring. In this way, no one chromosome can overpower the population in a single generation, and no matter how close the actual fitness values are, there is always constant pressure to improve. While the problem of premature convergence still exists, it is greatly reduced by allowing other chromosomes a chance to mix information with high fitness chromosomes. The disadvantage of using ranking is speed. In not allowing better chromosomes to guide the population easily, good answers are slower to develop.

## 2.5 Breeding

The previous step, reproduction, created a population whose members currently best solve the problem. However, many of the chromosomes are identical and none are different than those in the previous generation. Breeding combines chromosomes from the population and produces new chromosomes that, while they did not exist in the previous generation, maintain the same gene pool. In natural evolution, breeding and reproduction are the same step, but in GAs they have been separated to allow different methods for each to be experimented with and independently evaluated. It is in this step where GAs can exploit knowledge of the gene pool by allowing good chromosomes to combine with chromosomes that aren't as good. This is based on the assumption that each individual, no matter how good it is, doesn't contain the answer to the problem. The answer is contained in the population as a whole, and only by combining chromosomes will the correct answer be found.

There are many methods used for breeding; with the most common being crossover. Crossover typically takes two chromosomes and swaps parts of each to create two new chromosomes. Many variations on crossover have been used, but no results have shown

before	after
001100	000010
110011	111101
↑ ↑	↑ ↑

Figure 2: Example of two-point crossover (crossover points indicated by arrows)

which is decisively better. The crossover the author used to train the product networks was a simple two-point crossover. Two random points are chosen in the chromosome, and the bitstring between the two points is swapped between the two chromosomes. An example is shown in Figure 2.

## 2.6 Mutation

The last step in creating a new generation is based on the assumption that while each generation is better than the previous, the individuals that die may have some information that is essential to the solution. It is also possible that the initial population didn't have all the necessary information. The reinjection of information into the population is called mutation. Again, there are many ways to implement mutation, but essentially all choose and change members of the population randomly.

The method the author used was to simply inject a constant number of mutations every generation. The number of mutations used was approximately 0.25% of the total number of bits in the entire population. These mutations were then randomly distributed among all the bits, with each bit having the same chance of mutating. A mutation involved a 50/50 chance of setting the bit to a 1 or 0, in effect giving the mutated bit a 50/50 chance of changing. This means that any specific chromosome may or may not mutate, with a small chance that it could severely mutate.

## 2.7 An Application

A product network was trained that calculates the optimum width of the transistors in a CMOS switch given temperature, power supply voltage, and minimum conductance as inputs. While there are many excellent analysis tools available, such as circuit simulators, there are almost no software packages available that transform performance specifications into a circuit schematic. This network is designed as an aid to CMOS circuit designers, and was first proposed by Thelen in [4].

The data used to train the network was extracted from several SPICE simulations with differing transistor dimensions, temperatures, and power supply voltages. In the training set created from this data, the voltages ranged from 3 to 12 volts, the temperature from 303 to 403 °K, and the transistor width from 2 to 20 micrometers. Using these inputs, the conductance could range from approximately 1 to 500 micro-mhos. Two hundred data points were collected and a sample from these points is shown in Table 1.

Voltage	Temperature ( $^{\circ}\text{K}$ )	Conductance	Desired Width
3	303	1.026E-6	2
3	303	3.806E-6	3
3	303	6.593E-6	4
3	303	1.204E-5	6
3	303	1.752E-5	8
3	303	2.851E-5	12
3	303	3.951E-5	16
3	303	6.152E-5	24

Table 1: Sample from the data points used to train the network

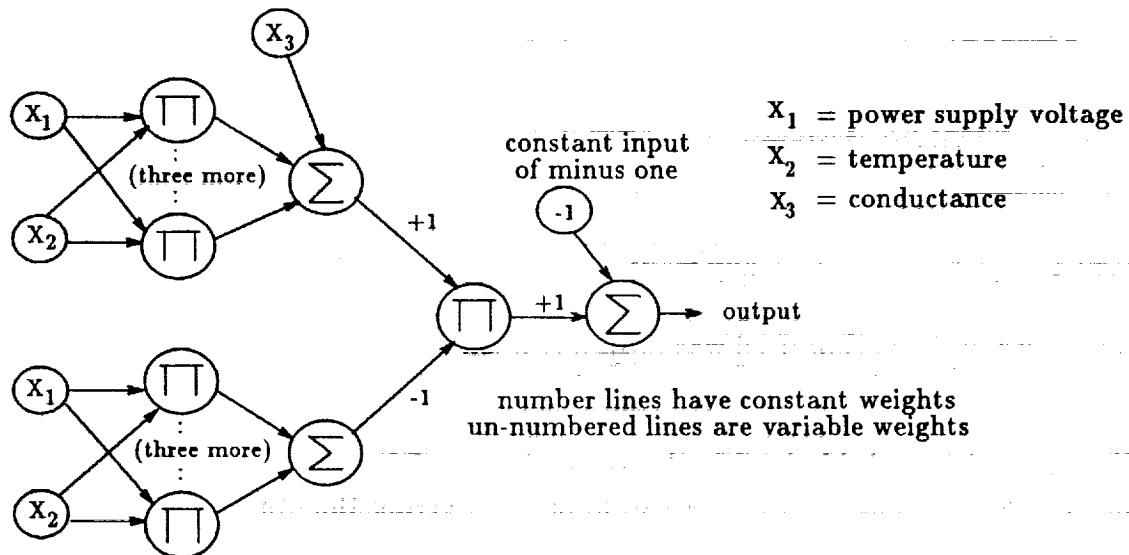


Figure 3: The product neural network trained to select the width of a CMOS switch

The configuration of the product network was designed by Thelen using *a priori* information about the equations to model a CMOS switch. The layout of the network is shown in Figure 3.

### 3 Results

The first attempts at training the product network had very consistent, but wrong, results. Through many runs of the GA, every solution represented a network that gave outputs of approximately ten for the transistor width, with no regard for the input.

The first success came when the population was seeded with an approximation to the solution. This approximation was derived by a curve-fitting program using the training data. When seeded, the GA was able to quickly improve the approximation and find a

network that gave the desired output. While seeding did indeed allow an answer to be found, it was desired that the GA could find an answer using an initial random population.

Better success was found using a penalty function. Penalty functions decrease the fitness of a chromosome by adding constrictions to the solution. The penalty subtracted from the fitness of a chromosome dependent upon how close the output of two consecutive data points were. The closer the two outputs for the two points were, the larger the penalty. With the addition of this penalty function, the GA was able to find a solution for the network given an initial random solution.

## 4 Discussion

The initial results were very surprising. The inability of the GA to find an appropriate solution meant that either the network could not solve the problem, or that the real solution to the problem was extremely difficult to find. Previous work by Thelan showed that indeed a solution to this problem did exist. This meant that the real solution must be difficult for the GA to find. In fact, when seeding the GA with approximate solutions, an answer was found.

There are three ways to make a problem difficult for a GA to solve. Either the solution space is extremely convoluted, the best solution occupies a very small portion of the solution space, or the solution space is misleading to a GA. Since proving whether a GA is being misled is very difficult, the other two possibilities were considered. Comparing the solutions found by the GA showed that they converged to the same answer each time. Thus, the solution space was not too convoluted for the GA to search.

The correct solution was not found with an initial random population. However, it was found with the insertion of the penalty function. (The effect of the penalty function was to place a pole in the middle of the unwanted solution, thus allowing the GA to continue searching the space, and find the correct solution.) This leads the authors to believe that the right answer occupied a very small portion of the solution space, allowing the GA to more easily find the undesired answer.

This example points out one common problem with GAs. In using GAs, often the solution space is not very well known, and suboptimal answers can often dominate the solution space. Indeed, if the problem to be solved is incorrectly or incompletely represented, the GA will take advantage of these mistakes, and produce wrong answers.

## 5 Conclusion

It has been shown that product networks can be successfully trained with Genetic Algorithms. A product network has been trained to give the width of CMOS switch, given power supply voltage, temperature and minimum conductance specifications for the switch. Further research will be done to compare the use of GAs to backpropagation in product networks. Also, the capabilities of product networks will be compared to traditional neural networks. While product units have been shown to have superior capabilities over

traditional summation units, almost no studies to compare different networks have been done.

## References

- [1] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1989.
- [2] R. Durbin and D. Rummelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, Vol 1, pages 133 - 142, 1989.
- [3] D.E. Rumelhart, G.E. Hinton, and J.L. McClelland. *Parallel Distributed Processing* 1, Chapter 8: Learning Internal Representations by Error Propagation, pages 318 - 362. Cambridge, MA, and London: MIT Press, 1986.
- [4] D. Thelen. A neural network for designing CMOS switches: an application for product units. In *JWSUUIISCNC '91: Proceedings of the 1991 Joint WSU/UI Interstate Student Conference on Neural Computation*, pages 100 - 109. Jack Medor, EE Dept, Washington State University, April 1991.
- [5] D. Whitley. Selective pressure and ranked based allocation. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116 - 123. Lawrence Erlbaum Associates, Inc., Hillsdale NJ, 1989.

This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

# VLSI Synthesis of Digital Application Specific Neural Networks

Grant Beagles  
Department of Electrical Engineering  
Montana State University  
Bozeman, Montana 59717

Kel Winters  
Advanced Hardware Architectures, Inc.  
Moscow, Idaho 83843

***Abstract-*** Neural networks tend to fall into two general categories, 1) software simulations, or 2) custom hardware that must be trained. The scope of this project is the merger of these two classifications into a system whereby a software model of a network is trained to perform a specific task and the results used to synthesize a standard cell realization of the network using automated tools.

## 1 Introduction

Neural net research may be roughly classified into two general categories; software simulations or programmable neural hardware [2,6].

Many neural network simulators are readily available. The major drawback to all of them is that, no matter how well written, they are run on a sequential machine. This means that the software must simulate the parallelism of the network and slows down dramatically as the number of connections increases [1,3].

Hardware neural networks are usually general purpose and must be trained. Depending on the training, a significant percentage of the total hardware resources may be unused. By defining the network with a software model and then synthesizing the network from that model, all of the silicon area will be utilized. This should result in a significant reduction in die size when comparing the application specific version to a general purpose neural network capable of being trained to perform the same task.

## 2 Network Modeling

The simulator that is being used for this project is version 2.01 of **NETS** written by Paul T. Baffes of the Software Technology Branch of the Lyndon B. Johnson Space Center [3]. This simulator was chosen for several reasons. **NETS** has a flexible network description format, the source code is available, and the weight matrix may be stored in an ASCII file for easy use in later steps.

As a first design effort, a simple numeral recognition network with three layers and 37 neurons was defined. The network consists of a 5 by 6 input layer, one hidden layer that is also 5 by 6, and a 1 by 7 output layer. This network is fully connected. Figure 1 contains the **NETS** description of the network.

```
LAYER : 0          --INPUT LAYER
      NODES : 30
      X-DIMENSION : 5
      Y-DIMENSION : 6
      TARGET : 2
```

```
LAYER : 1          --OUTPUT LAYER
      NODES : 7
      X-DIMENSION : 1
      Y-DIMENSION : 7
```

```
LAYER : 2          --FIRST HIDDEN LAYER
      NODES : 30
      X-DIMENSION : 5
      Y-DIMENSION : 6
      TARGET : 1
```

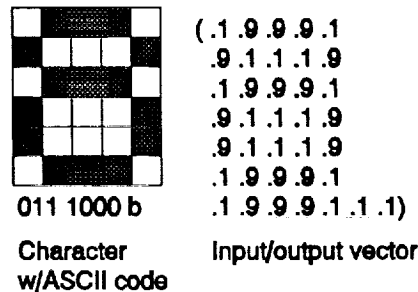
**NETS description of neural network.**

**Figure 1.**

A training set consisting of ten digits (0 through 9) and the corresponding ASCII values is used to build the network weighting matrix. Figure 2 illustrates a typical character representation and its corresponding input/output vector. The network training set does not include any noisy or corrupted data to simplify the model. Training the network required 100 iterations and was completed in about 6 minutes. The fully connected network has 1110 connections. The weights in the weight matrix range between  $\pm 1.7$  following training.



Once the network is trained, the number of connections is reduced. This is done by setting all weights having an absolute value less than a specified value to zero (no connect). This process is easily automated allowing various cut off values to be evaluated. The modified weight matrix is evaluated using NETS to determine whether or not the network will still satisfactorily perform its designed task. Table 1 summarizes the results of reducing the network.



Example of training set element.  
**Figure 2.**

CUT-OFF VALUE	NUMBER OF CONNECTIONS	SATISFACTORY PERFORMANCE	THRESHOLD <sup>1</sup>
0.3	688	yes	0.5
0.4	530	yes	0.5
0.5	413	yes	0.5
0.55	344	yes	0.5
0.6	288	no	----

<sup>1</sup> Any value  $\geq$  threshold is a "one" otherwise "zero".

**Table 1.**

The actual cut-off values tested ranged up to 1, however, all results with a cut-off above 0.55 were inconsistent with the desired results. Figure 3 is the test vector for the character shown in figure 2 with its associated output vector. (The cut-off is 0.55 and the threshold is 0.5.)

### 5.3.4

-- test set for min.net

(.1 .9 .9 .9 .1-- "8"

.9 .1 .1 .1 .9

.1 .9 .9 .9 .1

.9 .1 .1 .1 .9

.9 .1 .1 .1 .9

.1 .9 .9 .9 .1)

Outputs for Input 8:

( 0.002 0.846 0.994 0.865 0.036 0.257 0.164)

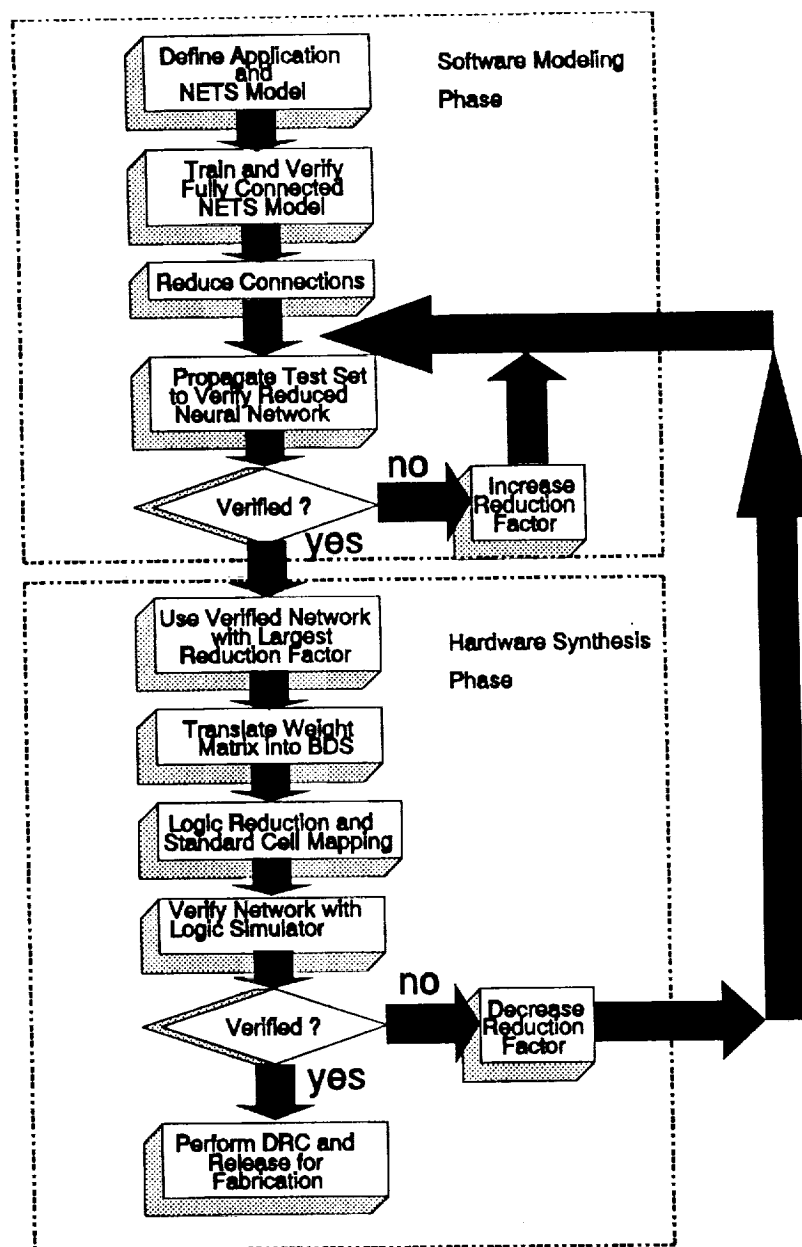
Output vector for test vector from figure 2.

Figure 3.

With the threshold value taken into consideration the output is 011 1000, which is the ASCII code for "8".

## 3 Logic Synthesis

The intent of the neural network synthesis process is to provide a fully automatic path to silicon realization once a network model has been constructed and verified in the **NETS** environment. The entire synthesis process is schematically shown in figure 4. The **OCT** tool set from the University of California, Berkeley [8], was chosen for the back end of this procedure, which includes logic optimization, technology mapping, standard-cell place-and-route, and composite artwork assembly and verification.



Application specific neural network synthesis process.

Figure 4.

First, the completed neural network topology is translated from the **NETS** environment to the **OCT** hardware description language **BDS** by a *NETS-to-OCT* program written for this purpose. A simple example of a single neuron in **NETS** netlist and the corresponding **BDS** description are shown in figure 5. The **BDS** file is then compiled into unminimized logic

### 5.3.6

functions by the OCT tool **Bdsyn**. These are mapped into a standard cell library by **MisII**. Currently, the **SCMOS2.2** standard-cell library from Mississippi State University is used, implemented in the **SCMOS6 N-Well CMOS** process available from the National Science Foundation **MOSIS** program. This process has a minimum feature size of two microns.

```
LAYER : 0--INPUT LAYER
NODES : 5
TARGET : 1
```

```
LAYER : 1--OUTPUT LAYER
NODES : 1
```

Majority logic NETS description.

```
MODEL dumb
    out<0>,sum0<4:0>=in<4:0>;

ROUTINE dumbnet;

!    target layer # 0 node # 0

sum0<4:0> = 8
           + in0<0>
           + in0<1>
           + in0<2>
           + in0<3>
           - in0<4>

IF sum0<4> EQL 1
    THEN out<0> = 1
    ELSE out<0> = 0;
ENDROUTINE;
ENDMODEL;
```

Majority logic BDS description.

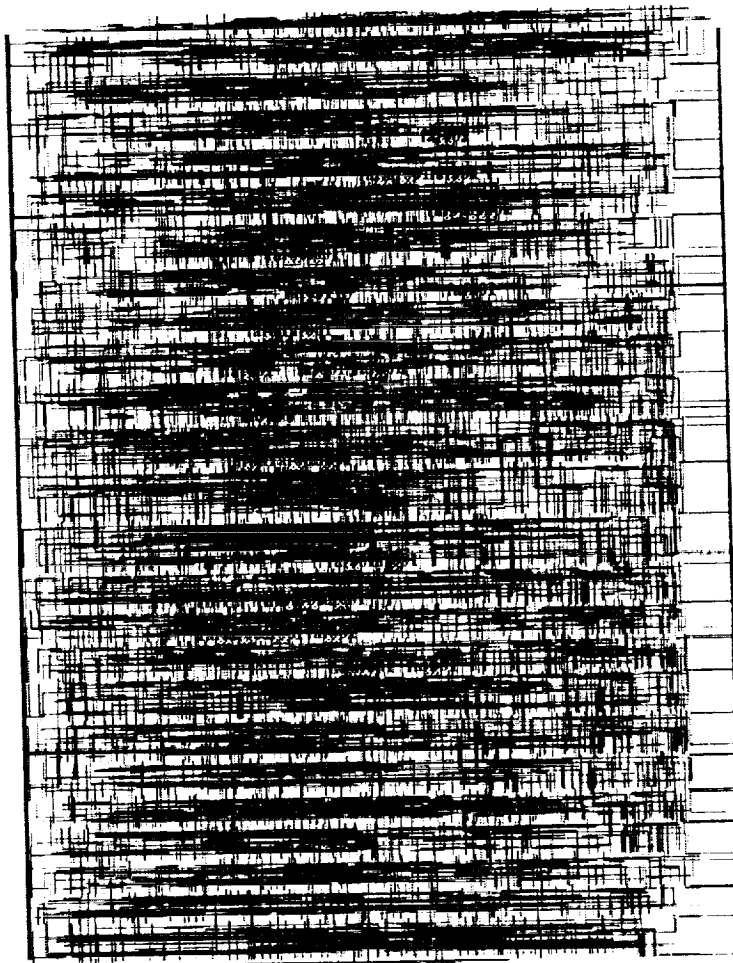
Figure 5.

**MisII** is an *n-level* logic optimizer, which creates a realization of a logic function from a given cell library minimizing both worst-case propagation delay and the number of cells required. The relative priority of area versus speed is user selectable. The result is stored in the OCT

database and may be verified with **MUSA**, a multilevel simulator in the **OCT** suite. From here, the design process may be easily iterated from the **NETS** description forward as shown in figure 4.

A number of additional **OCT** tools are available for padding composition, composite placement and channel routing, power distribution routing, and artwork verification. Artwork may be generated from the **OCT** database in Caltech Intermediate Format (**CIF**) for release to **MOSIS** or other foundry services.

The standard-cell realization of the digit recognizer described previously is shown in figure 6. Its 37 neurons required 2741 standard cells in 47 square millimeters.

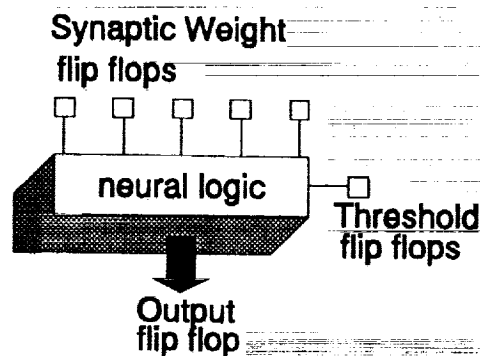


Standard cell realization of character recognizer.

**Figure 6.**

## 4 Conclusions and Future Directions

Figure 7 shows a block diagram of a 5 input programmable neuron. To build the digit recognizer using this generic neuron would require about 45 neurons. The actual network has 37 neurons. The increased number of generic neurons is due to the five input limitation. Many of the nodes in the network have more than five inputs. With the generic neurons, multiple neural cells would be connected at the outputs giving the behavior characteristics of a neuron having a larger number of inputs. The number of standard cells required for the entire network realized with the generic 5 input neuron is approximately 8280 (~45 neurons by 184 standard cells per neuron [7]). This network would cover nearly 141 square millimeters.



Generic five input neural cell.  
Figure 7.

As stated previously, the network created with the methodology described here requires 2741 standard cells and 47 square millimeters. This represents a 66% reduction in the number of cells used and silicon area. This reduction will allow the chip to be fabricated at a significantly lower cost than a chip with a sufficient number of the generic neurons. Furthermore, all of the silicon area in the application specific area is utilized whereas, a significant percentage is unused in the general model. These results are very preliminary. Experiments with simpler models suggest that substantial improvements in standard cell optimization remain possible.

The models used in this research were trained using ideal training sets, meaning that the characters were well formed and the level of contrast between the background and the characters was high. For the neural network to have any real value, a larger training set would be necessary. This set would have both poorly formed and low contrast examples of each character. Using a training set of this type would cause an increase in the number of connections necessary in the network [5].

The synthesis process described may be used to deliver an application specific neural network, trained to perform a specific task at less cost than utilizing general neural hardware. Silicon area will be more highly utilized in the application specific case since only the necessary circuitry is fabricated. Although more research is necessary, early results show the method to be promising.

## Acknowledgement

This work was supported by an educational grant from the National Science Foundation MOSIS program and equipment donations from the Hewlett-Packard Co, Tektronix Inc., and Advanced Hardware Architectures Inc. The authors would especially like to thank Paul Cohen, of Advanced Hardware Architectures; Andrea Casotto of UC Berkeley; Dr. Gary Harkin, Jaye Mathisen, Diane Mathews, and Bob Wall, of Montana State University; and Dr. Gary Maki, of the University of Idaho; for their invaluable assistance.

## References

- [1] H. C. Anderson, "Neural Network Machines," *IEEE Potentials*, Vol. 8 no. 1, pp. 13-16, Feb 1989.
- [2] J. A. Anderson, D. Hammerstrom, and L. D. Jackel, "Neural Network Applications for the 90's," *IEEE Videoconference*, May 23, 1991.
- [3] P. T. Baffles, *NETS User's Guide*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX.
- [4] M. W. Firebaugh, *Artificial Intelligence*, Ch. 18, PWS-KENT, Boston MA, 1988.
- [5] H. P. Graf, L. D. Jackel, W. E. Hubbard, "VLSI Implementation of a Neural Network Model," *IEEE Computer*, pp. 41-49, March 1988.
- [6] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, Vol 1 and 2, MIT Press, Cambridge, MA, 1986.
- [7] S. Wandler and D. Metcalf, "Development of a Neural Network Integrated Circuit," Senior Project Report, Montana State University, Department of Electrical Engineering, 1991.

### 5.3.10

- [8] A. Casotto, ed., *OCTTOOLS Revision 5.1 User Guide*, University of California Berkeley, Electronics Research Laboratory, Berkeley, CA, 1991.



## Measurement Selection For Parametric IC Fault Diagnosis<sup>1</sup>

A. Wu and J. Meador

School of Electrical Engineering and Computer Science

Washington State University

Pullman, Washington 99164-2752

meador@eecs.wsu.edu      awu@eecs.wsu.edu

(509) 335-5363      FAX: (509) 335-3818

**Abstract-** This paper describes experimental results obtained with the use of measurement reduction for statistical IC fault diagnosis. The reduction method used involves data pre-processing in a fashion consistent with a specific definition of parametric faults. The effects of this preprocessing are examined.

### 1 Introduction

An integrated circuit test is specified by a combination of input and output signals which characterizes some attribute of ideal circuit function. The presence of faults in a fabricated circuit will cause observed output signals to deviate from the simulated ideal. A fault diagnostic is a decision rule combining what is known about an ideal circuit test response with information about how the response is distorted by fabrication variations and measurement noise. The rule is used to detect fault existence in fabricated circuits using real test equipment.

The IC failure diagnosis problem can be viewed as a statistical pattern recognition problem. Instead of extracting output response parameters explicitly and comparing with the specification, the output responses can be identified into faulty or non-faulty according to some classification decision rules. It has been positively demonstrated that pattern classification technique can be used in IC diagnosis [Mea90].

Recent experiments [Mea91] have showed that feedforward network classifier (FFN) generally perform as well as or even better either than the traditional statistical parametric classifier, Gaussian Maximum Likelihood Classifier (GML) or the non-parametric classifier, the K-nearest Neighbors classifier (KNN). However, it usually needs more computational efforts for FFN in the training phase to establish the discriminant function. To be more effective, there is a need to find ways to consistently reduce this training overhead, while simultaneously retaining prediction accuracy.

Nevertheless, performance of a classifier depends on the data presented in the training, the discriminant function established in the training phase as well as the classification algorithm of the classifier. To ensure high performance accuracy, essential information has to be presented in the training data for the establishment of the discriminant function of the classifier.

---

<sup>1</sup>This work supported by NSF-UIC CDADIC Project 90-1.

In IC diagnosis, determination of a circuit fault is to classify the circuit from the input-output measurement according to a decision rule which is built upon the estimated prior probability distribution in the performance space of the circuit. Judgment is made according to the decision rule of the classifier established in training, which defines the decision boundaries for the classification. For accurate classification, decision boundaries of the classifier has to be coincided with or close to the performance specification criteria or boundaries. Such decision boundaries has to be captured by the classifier to set up the discriminant function or decision rules in training. The acceptance region of the fabricated circuits lines between the upper and lower performance specification limits. For highest accuracy, in the training phase, the decision boundaries have to be built around the specification transition space. In this case, they are the upper and lower specification limits instead of the mean of the performance distribution.

It is a well known fact that, in back-propagation training algorithm, input values are multiplied by the derivative of the logistic function, such that, a window is placed on the current estimated decision boundaries, not the mean [Lip88]. This character is very important in IC diagnostic problem especially the go/no go testing. If the discriminant function is established around the specification boundaries, it will improve the performance of the classifier. Besides, training on these boundaries, it will improve the training computational load.

To improve the training effort, it is therefore logical to train a FFN based on the decision boundaries. If data used in the training is collected around these boundaries, the discriminant function computed by the trained network will be more accurate in these regions. It will improve the training computational load as well, since fewer epochs are required to converge to a given accuracy. This paper reports on experiments conducted to help verify this idea.

## 2 Data Reduction Method: Boundary Band Data Pre-processing

In contrast to the design task, the concern of IC fault diagnosis is mainly on whether the circuit performance fall within the acceptance region instead of the performance mean. In other words, the specification transition boundaries are the most concerns in IC diagnosis. If the decision rules or decision boundaries of any diagnosis algorithm are based on these transition boundaries, it is reasonable to expect a highly accurate and effective diagnostic capability.

As discussed in the preceding section, there is a need to improve the computational load of training FFN classifier even though it has a better diagnostic capability than the other traditional statistical classifiers. Here, we proposed a Boundary Band Data (BBD) training method for FFN training to improve the computational load in the training phase. The essence of the proposed method is based on the characteristic of FFN. In back-propagation training algorithm of FFN, input values are multiplied by the derivative of the logistic function, such that, a window is placed on the current estimated decision boundaries, not

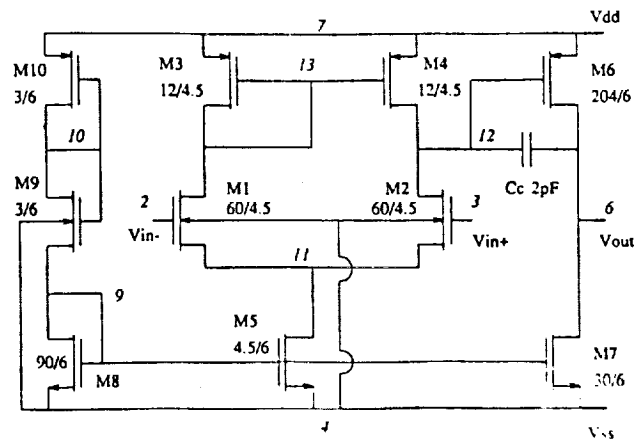


Figure 1: Operational amplifier circuit diagram

the mean. If the discriminant function or decision boundary of the trained network is set up around the specification transition boundaries, it will improve the performance of the classifier. Besides, training on these boundaries, it will improve the training computational load. Making use of this distinctive characteristic of FFN, the proposed Boundary Band Data training method is to train a FFN with those data gather from the proximity of performance specification transition boundaries.

### 3 Experiment Set Up

To investigate the feasibility of using BBD in FFN training, experiments were conducted in this study. The transient response and frequency response of the operational amplifier shown in Figure 1 were used for the experiments. For frequency response experiments, the open loop frequency response of the operational amplifier were used. For transient response experiments, the step response of an inverting amplifier with the same operational amplifier for the frequency response experiments are used. The circuit configuration for the transient response experiment is shown in Figure 2.

### 4 Fault Definition

All experiments were designed to detect parametric faults in an operational amplifier. Monte Carlo simulation of MOSFET model parameters was used. Only those statistical independent model parameters were used so that the correlation effect among model parameters was eliminated. In each of the experiments, circuit fault was defined as a large variation in one of the independent model parameter. In our experiments, three types of parametric faults were used. They were variations in MOSFET oxide thickness (of all the transistor in the circuit), zero bias threshold voltage, and junction depth.

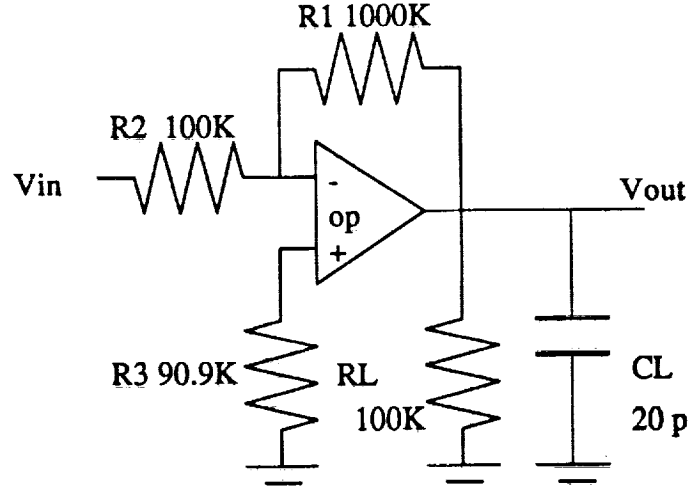


Figure 2: Inverting amplifier circuit configuration

Monte Carlo simulation using SPICE with a large variation around the mean value in the chosen model parameter was used. With a pre-selected performance criteria, the appropriate upper and lower limits of the model parameter which defined the fault and normal transition boundaries could be determined. For example, we were interested to define the fault and normal boundaries for the experiment with the inverting amplifier circuits. The fault was chosen to be variation in oxide thickness. So the SPICE model parameter,  $tox$ , was chosen with mean value equal to  $600\text{\AA}$ . The related circuit performance criteria were the step response overshoot and the slope of the stem response. Using the above method, the transition boundaries were set at  $400\text{\AA}$  and  $800\text{\AA}$ . The acceptance region was set between these limits. Any circuit fell within this region was defined to be normal; otherwise, it was defined as faulty.

In this study, three types of parametric faults were studied. They were circuit faults in oxide thickness, junction depth, and zero biased threshold voltage. For each of the experiments, there was only a single parametric fault existing in the circuit. The mean model parameter values for normal circuit and transition boundaries for circuit faults are listed in Table 1.

## 5 Experiment Description

Eight experiments which were divided into two categories were investigated for the BBD training methods. For the first category, it consisted of six experiments. To simplify the problem, in each of the experiments, only one SPICE model parameter was allowed to alter. It was under the assumption that there was no process existing in IC fabrication except the process fault. Even though such assumption might not be realistic for actual IC fabrication, the goal of these non-noisy experiments was to study the effect of the proposed BBD training method under the ideal condition. In these non-noisy experiments, the

Experiment	Parametric Fault	Performance parameters	Mean	Upper Limit	Lower Limit
1	oxide thickness	slew rate and overshoot	600Å	800Å	400Å
2	junction depth	slew rate	0.4μm	0.6μm	0.2μm
3	threshold voltage	slew rate	0.7V	0.5V	0.9V
4	oxide thickness	slew rate	600Å	800Å	400Å
5	junction depth	slew rate	0.4μm	0.6μm	0.2μm
6	threshold voltage	slew rate	0.7V	0.5V	0.9V
7	oxide thickness	slew rate and overshoot	600Å	800Å	400Å
8	junction depth	slew rate	0.4μm	0.6μm	0.2μm

Table 1: SPICE model parameter mean and transition boundary values

transient response of the amplifier in a closed loop inverting circuit under various nominal and faulty conditions was used to develop the experiment database for the experiment 1 to 3. For experiments 4 to 6, open loop frequency response of the amplifier under the same nominal and faulty conditions as in experiments 1 to 3 were used. The six non-noisy experiments were:

*Exp.1:* Detect a 33% variation in oxide thickness of all the transistor in the circuit by observing the circuit open-loop frequency response.

*Exp.2:* Detect a 50% variation in junction depth of all the transistor in the circuit by observing the circuit open-loop frequency response.

*Exp.3:* Detect a 30% variation in threshold voltage of all the transistor in the circuit by observing the circuit open-loop frequency response.

*Exp.4:* Detect a 33% variation in oxide thickness of all the transistor in the circuit by observing the circuit time-domain step response.

*Exp.5:* Detect a 50% variation in junction depth of all the transistor in the circuit by observing the circuit time-domain step response.

*Exp.6:* Detect a 30% variation in threshold voltage of all the transistor in the circuit by observing the circuit time-domain step response.

The second categories of the experiments consisted of two experiments which were similar to experiment 4 and 5 with the difference that there were process noise existed. It was under the assumption that there were process noises in the fabrication but not contributed to circuit faults. Such assumption was more realistic for actual IC fabrication. The goal of these experiments was to study the effect of BBD training of FFN under the non-ideal environment. Those process noises were generated by varying those statistical independent model parameters [She88] of lateral diffusion (LD), substrate doping density (NSUB), bulk threshold parameter ( $\gamma$ ), and channel-length modulation ( $\lambda$ )

at most one percent. In these two experiments, the transient response of the amplifier in a closed loop inverting circuit under various nominal and faulty conditions was used to develop the experiment database. For these noisy experiments, only one parametric fault were assumed but accompanied with all the process noises listed above. The noisy experiments were:

*Exp.7:* Detect a 33% variation in oxide thickness of all the transistor with process noise in the circuit by observing the circuit time-domain step response.

*Exp.8:* Detect a 50% variation in junction depth of all the transistor with process noise in the circuit by observing the circuit time-domain step response.

## 6 Data Generation

In each of the experiments, two data distributions namely normally distributed data and boundary band data as shown in Figure 3 and 4, were used to build up the experiment database. 120 simulated responses were obtained via a Monte Carlo simulation for each data distribution. Data for the boundary band distribution were generated around the transition boundaries. The sample data distribution of each of the experiment is similar to Figure 3 and 4 with difference in variation percentage of the corresponding model parameter. And the corresponding circuit performance distribution from the two data distributions were showed in Figure 5 and 6. 60 of the responses correspond to the fault free condition and 60 correspond to the faulty condition. In other words, there were four set of data consisting the experimental database for each of the experiments. The data sets were data for faulty circuit with normal distribution, data for normal circuit with normal distribution, data for faulty circuit with boundary band distribution, and data for normal circuit with boundary band distribution. For a particular data distribution, 30 responses from each class (normal/faulty) were used for classifier training. After training, classifier were tested on the unseen data from the trained data distribution as well as the data from the other type of distribution.

## 7 Classifier Training

As mentioned in the introduction, the objective of this study is to contrast the effectiveness of a feedforward network classifier trained on boundary band data against that of traditional statistical classifiers trained on normally distributed data and feedforward network as well in the context of IC fault diagnosis. Classifiers used in this study were Gaussian Maximum Likelihood Classifier, K-Nearest Neighbor Classifier and Feedforward Classifier.

Thirty patterns chosen from each the normal and faulty class of each of the experimental database for the training. For GML, training data was used to build the corresponding mean matrix, covariance matrix and the inverse of covariance matrix. For KNN, training data was used as the base for the classifier. For FFN, different types of training were used

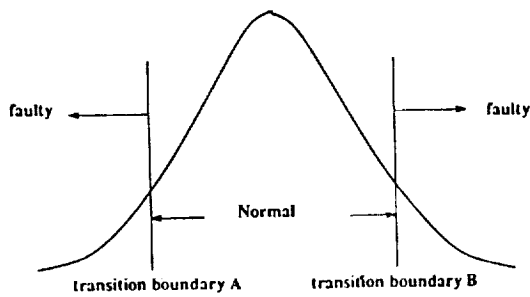


Figure 3: Data with normal distribution

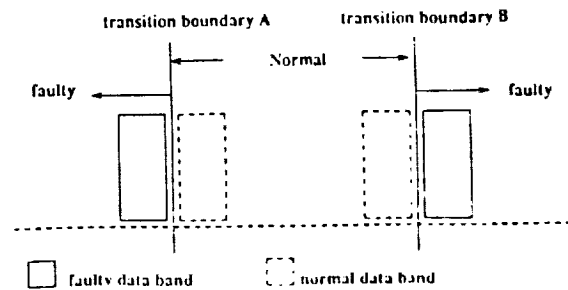


Figure 4: Data with boundary band distribution

to establish the discriminant function for the corresponding FFN. There were FFN trained with non-noisy normal distributed data, non-noisy boundary band data, noisy normal distributed data, and noisy boundary band data. For each trained FFN, only one of the listed training method was used. Unlike traditional statistical classifiers, there are some training criteria can be chosen. We trained our FFN based on the total sum of square error of all the training data for a particular type of training or up to a preset training epoch limit.

## 8 Classifier Computational Load Calculation

The performance of each classifier was not only measured in terms of predictive accuracy on previously unseen data, but also the number of floating point operations (FLOPS) required to construct the classifier, and the number of FLOPS required to perform a diagnostic classification. Number of FLOPS computed for each of the classifier of each experiment is based on the implementation algorithm. It is not the actual computer operation. Since different software packages are used in the implementation of the classifier, it is not accurate if they are compared based on the real CPU time. In comparing computation requirement in testing, number of flops required per pattern are calculated with the equations  $2y(n + 1) + 2z(y + 1)$  for Feedforward network,  $mn(3 + 2n)$  for Gaussian Maximum Likelihood classifier,  $3mnp$  for K-Nearest Neighbor classifier. ( $m$ : no. of class,  $n$ : no. of measurement for each pattern,  $y$ : no. of hidden unit,  $z$ : no. of output unit,  $p$ : no. of pattern for the training set in each class)

## 9 Experiment Results

In each of the experiments, the performance of the classifiers were evaluated for the prediction accuracy of unseen data as well as the training and testing computational load. The results of non-noisy experiments 1 to 6 are summarized in Table 2 and Table 3. The results of noisy experiments 7 and 8 are summarized in Table 4.

Classifier	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6
Accuracy (unseen data - %)						
FFN(Norm)	82.2	88.8	97.7	88	95.5	98.8
FFN(Boun)	99.4	97.2	98.8	97.2	98.8	96.6
FFN(ReBo)	98.8	97.7	100	98.3	99.4	97.2
GML	N.W	N.W	N.W	50	N.W	N.W
1NN	85	96	98.3	50	96	100
3NN	81.6	100	98.3	53.2	100	100
5NN	80	96	98.3	50	96	100
Setup FLOPs (total)						
FFN(Norm)	3e8	3e8	3e8	2.7e8	2.7e8	2.7e8
FFN(Boun)	3e8	3e8	3e8	2.7e8	2.7e8	2.7e8
FFN(ReBo)	6e7	6e7	3e7	5.4e7	5.4e7	2.7e8
GML	N.A	N.A	N.A	5e4	N.A	N.A
KNN	none					
Diagnostic FLOPs (per pattern)						
FFN(Norm)	1.5e3	1.5e3	1.5e3	1.4e3	1.4e3	1.4e3
FFN(Boun)	1.5e3	1.5e3	1.5e3	1.4e3	1.4e3	1.4e3
FFN(Rebo)	1.5e3	1.5e3	1.5e3	1.4e3	1.4e3	1.4e3
GML	N.A	N.A	N.A	5.6e3	N.A	N.A
KNN	1.5e4	1.5e4	1.5e4	1.3e4	1.3e4	1.3e4

N.A: Not applicable

N.W: Not working for the case due to singular covariance matrix

GML: Gaussian Maximum Likelihood Classifier

KNN: K-Nearest Neighbors Classifier

FFN(Norm): Feedforward network trained with normal distributed data

FFN(Boun): Feedforward network trained with boundary band data

FFN(ReBo): Feedforward network trained with boundary band data and reduced training epoch

Table 2: Classifier Accuracy and Computational Overhead of Exp. 1 to 6



We trained three FFN for each of the experiments. There was FFN trained with normal distributed data. In this training, the FFN (Norm) was trained up to a point that the *tss* did not changed much with ongoing training. We determined the training stopping point to be  $2e5$  epochs. FFN (Boun) was a FFN trained with the boundary band data around the transition values. The same training stopping point was used as in FFN(Norm). The third type of training used in these six experiments was a FFN trained with boundary band data but with fewer training epochs. The stopping point of this training method depended on the prediction accuracy of the trained FFN. We stopped the training whenever the prediction accuracy of the trained FFN was similar to the FFN(Boun).

From the results of the experiments 1 to 6, summarized in Table 2, it shown that feedforward networks had better performance than GML and KNN. Besides, in general, FFN trained with boundary band data had better prediction accuracy than that of FFN trained with normally distributed data. These results were observed as predicted in the proposed method section. It was because the decision boundaries of the trained networks were expected to set around the transition boundaries in the performance space. Moreover, trained with fewer epoch, in general, it had a better prediction accuracy. It was because of the nature of neural network. With fewer training, it might eliminate the trained network from memorize the training data. In most of the cases, inverse covariance matrix of the training data for GML could not be computed without further data preprocessing.

To investigate the effectiveness of boundary band training, different types of training were used in our experiment. There were FFN trained with non-noisy normal distributed data, non-noisy boundary band data, noisy normal distributed data, and noisy boundary band data. For each trained FFN, only one of the listed training method was used. The results of FNN trained with these method of the non-noisy experiments are summarized in Table 3. Experiments with process noises are summarized in Table 4.

Results shown in Table 3 and 4 were the prediction accuracy of FFN with different training methods. Each trained FFN was tested on the unseen data from both of the normal distributed database and boundary band database. As showed in Table 3 and 4, there were two prediction accuracy for each of the trained FFN which tested on unseen data from normal distributed database (labeled Normal) and from boundary band database (labeled Boundary). It showed that the boundary band training did work on both non-noisy and noise cases. In general, with fewer training epochs, FFN trained with boundary band data performed as well as and even better in some case than FNN trained with normal distributed data. And, there were a large training epochs and prediction trade off of FNN trained with boundary band data. Using this method, there was very few prediction degradation but with a significant reduction in computation load spending on training.

Frequency Response	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6
<b>Accuracy(unseen data-%) with various distribution</b>						
<b>FFN(Norm) trained with normally distributed data</b>						
Normal	95	96.6	98.3	96.6	96.6	98.3
Boundary	75.8	85	97.5	85	95	99.1
<b>FFN(Boun) trained with boundary band data</b>						
Normal	99.1	98.3	99.1	96.6	98.3	96.6
Boundary	100	95	99.1	98.3	100	100
<b>FFN(ReBo) trained with boundary band data and reduced training epoch</b>						
Normal	98.3	97.5	100	98.3	99.1	97.5
Boundary	100	98.3	100	98.3	100	96.3
<b>Setup Computation Load (Training)</b>						
<b>FFN(Norm) trained with normally distributed data</b>						
Epoch	2e5	2e5	2e5	2e5	2e5	2e5
Flops	3e8	3e8	3e8	2.7e8	2.7e8	2.7e8
Load	100%	100%	100%	100%	100%	100%
<b>FFN(Boun) trained with boundary band data</b>						
Epoch	2e5	2e5	2e5	2e5	2e5	2e5
Flops	3e8	3e8	3e8	2.7e8	2.7e8	2.7e8
Load	100%	100%	100%	100%	100%	100%
<b>FFN(ReBo) trained with boundary band data and reduced training epoch</b>						
Epoch	4e4	4e4	2e4	4e4	2e4	2e4
Flops	6e7	6e7	3e7	5.4e7	2.7e7	2.7e7
Load	20%	20%	10%	20%	10%	10%

Table 3: Comparison of Feedforward Network with Different Training of Exp.1 to 6

Transient Response	Exp. 7	Exp. 8
Accuracy (unseen data-%) with various distribution		
FFN(Norm) trained with normally distributed data		
Normal	98.3	95
Boundary	94.1	94.1
FFN(Boun) trained with boundary band data		
Normal	97.5	99.1
Boundary	98.3	96.6
FFN(ReBo) trained with boundary band data and reduced training epoch		
Normal	99.1	96.6
Boundary	98.3	95
Setup Computation Load (Training)		
FFN(Norm) trained with normally distributed data		
Epoch	2e5	2e5
Flops	2.7e8	2.7e8
Load	100%	100%
FFN(Boun) trained with boundary band data		
Epoch	2e5	2e5
Flops	2.7e8	2.7e8
Load	100%	100%
FFN(ReBo) trained with boundary band data		
Epoch	4e3	4e3
Flops	5.4e6	5.4e6
Load	2%	2%

Table 4: Comparison of Feedforward Network with Different Training of Noisy Exp.7 and 8

## 10 Conclusion

We studied the effectiveness of a feedforward network classifier trained on boundary data against that of traditional statistical classifiers trained on normally distributed data and feedforward network as well in the context of IC fault diagnosis. Eight experiments with and without process noises were conducted. In this study, experiment results once again demonstrated, in general, that feedforward network out performed the traditional statistical classifiers namely Gaussian Maximum Likelihood classifier and K-Nearest Neighbor classifier. Feedforward networks trained with boundary band data, it reduced the training effort with only little prediction degradation. Experiment results showed that the proposed boundary band data did improve the computational load needed for feedforward network training.

## References

- [1] William Y. Huang and Richard P. Lippmann, Comparisons Between Neural Net and Conventional Classifiers, *Proc. International Joint Conference of Neural Network*, pp. IV-485-489, 1988.
- [2] T. Lin, H. Tseng, A. Wu, N. Dogan and J. Meador, Neural Net Diagnostics for VLSI Test, *Proc. Second NASA SERC Symposium on VLSI Design 1990*, pp. 6.1.1-6.1.11, 1990.
- [3] J.L Meador, A. Wu, C.T. Tseng and T.S. Lin, Mixed Signal IC Test, *NSF Center for Design Analog-Digital Integrated Circuit Technical Report 90-1*, Project # CDADIC 90-1, Jan 1991.
- [4] Bing. J. Sheu, Chung-Ping Wah, Chih-Ching Shih, Wen-jay Hsu and Ming C. Hsu, Determination of Process-Dependent Critical SPICE Parameters for Application-Specific ICs, *IEEE Proc. International Conf. on Microelectronic Test Structure*, pp.73-78, 1988.

## Fuzzy Control of Magnetic Bearings

J. J. Feeley, G. M. Niederauer, and D. J. Ahlstrom

Department of Electrical Engineering

NASA Space Engineering Research Center for VLSI Design

University of Idaho

Moscow, Idaho 83843

**Abstract-** This paper considers the use of an adaptive fuzzy control algorithm implemented on a VLSI chip for the control of a magnetic bearing. The architecture of the adaptive fuzzy controller is similar to that of a neural network. The performance of the fuzzy controller is compared to that of a conventional controller by computer simulation.

### 1 Introduction

Magnetic levitation is receiving increasing attention as a viable alternative to conventional methods of moving and positioning objects [1]. NASA, for example, has developed a cryogenic cooler that uses magnetic bearings and actuators exclusively [2]. One of the more difficult aspects of the application of magnetic bearings is the control of the position of the shaft in the bearing housing. Considerable attention has been given to this problem recently. Williams et. al. [3] reported on the digital control of active magnetic bearings and showed how the flexibility of digital control was extremely useful in implementing a number of control algorithms including second-derivative and integral feedback. Chen and Darlow [4] describe an analog control system for an active magnetic bearing that uses velocity and acceleration observers to improve damping and cancel imbalance and other disturbance forces to greatly improve the overall system performance. Keith et. al. [5] discuss the magnetic support of flexible shaft at speeds up to 14,000 RPM using a PC-based digital controller implementing a proportional-derivative control algorithm. A comparison with an earlier analog proportional-derivative controller is also made. Chen [6] describes an active magnetic bearing control scheme using three parallel feedback loops to achieve dynamic stiffness, static stiffness, and damping. He presents a closed-form solution for controller parameters in terms of desired stiffness and damping. Humphris et. al. [7] present a comprehensive treatment of the active magnetic bearing control problem and compare the relative performance of low bandwidth and high bandwidth controllers. Scudiere et. al. [8] used a Texas Instruments TMS32010 digital signal processor to implement a proportional-integral-derivative control algorithm to successfully control the position of a number of small spheres and rotors. Feeley et. al. [9] described root locus design of a double lead-lag controller mapped into an equivalent digital controller via the Tustin transformation. The resulting algorithm has been implemented on an Intel 80KC196C microprocessor and used to control an analog computer model of the NASA magnetic bearing.

The difficulty of the control problem stems from two basic causes. The first is due

to the physical nature of the magnetic bearing system itself. As shown in Section 2, the uncontrolled magnetic bearing system is unstable, uncertain, and highly nonlinear. The instability is due to the relentlessness of gravity in causing any suspended object to fall. The uncertainties arise from the difficulties in modeling viscous friction, eddy currents, leakage flux, and accounting for disturbance forces due to vehicle acceleration, motion of the shaft, and other random events. The nonlinearities arise in the square-law nature of magnetic forces, the nonlinear relationship between actuator current and magnetic flux, and the nonlinear properties of materials in the magnetic circuit. The second basic cause of difficulty in the control problem stems from the decision to use digital control. Sampling is inherent in digital control and it is reasonable to expect poorer performance from a digital control system using data samples than from its ideal analog equivalent using continuous data. This inevitable degradation in performance encountered in moving from analog to digital control must be compensated for by the use of more sophisticated digital control algorithms and the other advantages inherent in digital control.

A control scheme that is effective in overcoming these two basic causes of difficulty in the control problem is presented in this paper. The scheme is based on the theory of fuzzy systems. The modeling problem is addressed by substituting the imprecise linguistic model of fuzzy theory for the precise model of physical theory. The sampling problem is addressed by implementing the fuzzy algorithm in a parallel architecture suitable for VLSI implementation thereby reducing processing time and allowing high sampling rates.

The remainder of the paper is organized as follows. Section 2 describes the magnetic bearing system and presents a mathematical model developed by Feeley et. al. [10]. In Section 3 some essential elements of fuzzy control theory are presented and an adaptive fuzzy controller is developed. In Section 4 the performance of the fuzzy controller is analyzed using a computer simulation based on the nonlinear model of Section 2. A adaptive fuzzy control VLSI chip architecture is outlined in Section 5 and some conclusions and recommendations are given in Section 6

## 2 Magnetic Bearing System

A schematic cross-sectional side view of NASA's magnetic bearing is shown in Figure 1 supporting one end of a rigid shaft. An end view would show the circular cross-section shaft centered in the annular gap created by the bearing housing and the shaft. Figure 1 also shows the shaft magnetic material inlays that provide paths for the magnetic flux produced by the adjacent bearing actuators. The actuators are symmetrically located in the bearing housing and consist of magnetic material pole pieces and coils of copper wire. A position sensor is located close to each actuator to measure the position of the shaft. A total of four actuator and position sensor assemblies are located at 90° increments around the circumference of the housing. Coordinated control of opposing actuators permits positioning of the end of the shaft anywhere in the annular gap. An identical bearing assembly supports the other end of the shaft. For simplicity, rotational forces are not directly accounted for and half of the shaft mass is assumed to be concentrated at the point of action

of the magnetic forces of each bearing assembly.

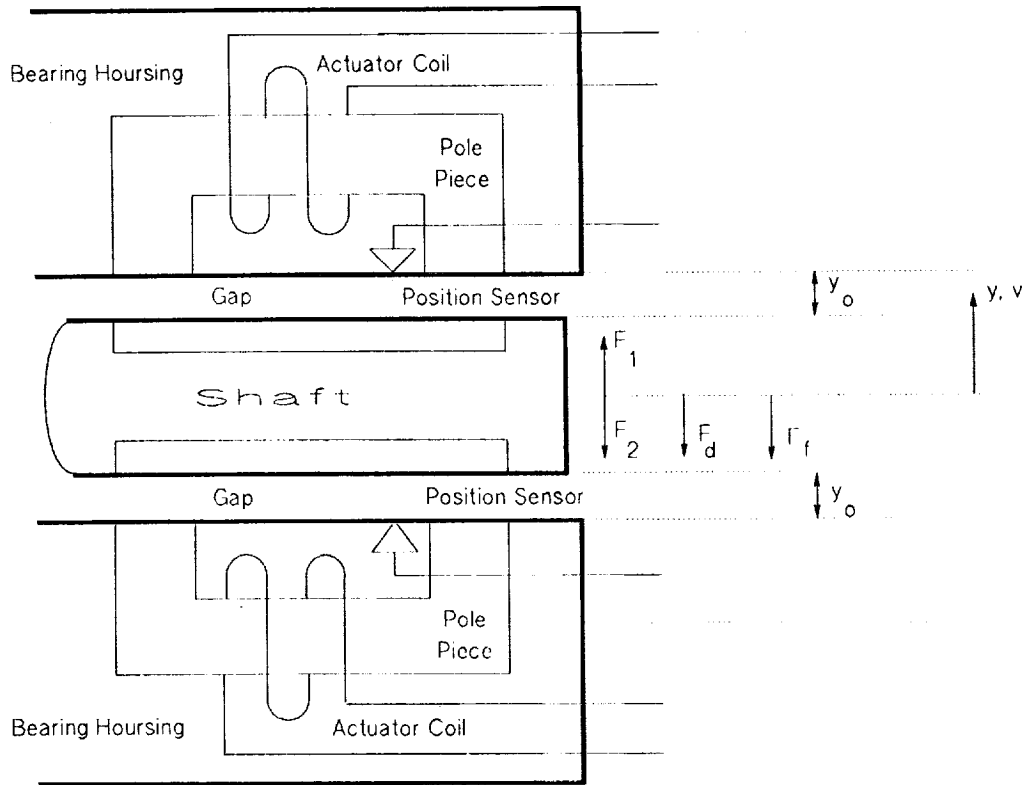


Figure 1: Schematic cross-section of magnetic bearing assembly

Assuming motion in the one-dimensional coordinate system defined in Figure 1, application of Newton's second law yields

$$\frac{d^2y}{dt^2} = \frac{F_1 - F_2 - F_d - F_f}{M}$$

where  $y$  is the position of the shaft,  $F_1$  is the magnetic force exerted by the upper actuator,  $F_2$  is the magnetic force exerted by the lower actuator,  $F_d$  is a disturbance force, and  $F_f$  is a viscous friction force.  $F_1$  and  $F_2$  are, in turn, defined by

$$F_1 = \frac{\mu_0 A}{4} \left[ \frac{N_c i_1}{y_0 - y} \right]^2$$

and

$$F_2 = \frac{\mu_0 A}{4} \left[ \frac{N_c i_2}{y_0 - y} \right]^2$$

where  $\mu$  is the magnetic permeability,  $A$  is the area of one pole face,  $N_c$  is the number of coil turns,  $i_1$  and  $i_2$  are the coil currents, and  $y_0$  is the initial gap distance. The friction force is assumed proportional to the square of the shaft velocity and is modeled mathematically as  $F_f = K_f v|v|$ , and the disturbance force is taken as an exogenous input.

The electromagnetic of the actuator are modeled with the aid of the circuit diagram of Figure 2. The circuit model consists of two loops, one for the primary coil current  $i_c$ , and a second for the induced eddy current  $i_e$ . Applying Kirchoff's voltage law to each loop yields the circuit equations

$$\begin{aligned} v_c &= Ri_c + N_c \frac{d\phi_c}{dt} + N_{ce} \frac{d\phi_e}{dt} \\ 0 &= R_e i_e + N_e \frac{d\phi_e}{dt} + N_{ec} \frac{d\phi_c}{dt} \end{aligned}$$

where  $v_c$  is the voltage applied to the coil,  $N_c$  is the number of turns in the coil,  $\phi_c$  is the flux produced by the coil current,  $N_{ce}$  is the number of turns of the coil linked by the flux produced by the eddy currents,  $\phi_e$  is the flux produced by the eddy currents,  $R_e$  is the resistance of the eddy current paths,  $N_e$  is the number of turns in the equivalent eddy current coil, and  $N_{ec}$  is the number of turns in the equivalent eddy current coil linked by the flux produced by primary current. Assuming the entire mmf drop of the magnetic circuit is taken across the two air gaps, the fluxes can be expressed in terms of the currents as  $\phi_c = \frac{\mu_0 AN_c i_c}{2y_{ag}}$  and  $\phi_e = \frac{\mu_0 AN_e i_e}{2y_{ag}}$  where  $y_{ag}$  is the distance between the pole piece and the shaft,  $y_0 - y$  for the upper gap and  $y_0 + y$  for the lower gap. Solving these equations for the time derivatives of the currents leads to

$$\begin{aligned} \frac{di_{c1}}{dt} &= - \left[ \frac{v}{y_0 - y} + \frac{R_c}{N_c L_1} \right] i_{c1} + \frac{k R_e}{L_1} i_{e1} + \frac{1}{N_c L_1} v_{c1} \\ \frac{di_{e1}}{dt} &= \frac{k R_c}{L_1} i_{c1} - \left[ \frac{v}{y_0 - y} + \frac{N_c R_e}{L_1} \right] i_{e1} - \frac{k}{L_1} v_{c1} \\ \frac{di_{c2}}{dt} &= - \left[ \frac{v}{y_0 + y} + \frac{R_c}{N_c L_2} \right] i_{c2} + \frac{k R_e}{L_2} i_{e2} + \frac{1}{N_c L_2} v_{c2} \\ \frac{di_{e2}}{dt} &= \frac{k R_c}{L_2} i_{c2} - \left[ \frac{v}{y_0 + y} + \frac{N_c R_e}{L_2} \right] i_{e2} - \frac{k}{L_2} v_{c2} \end{aligned}$$

where  $L_1 = \frac{\Delta \mu_0 A}{2(y_0 - y)}$ ,  $L_2 = \frac{\Delta \mu_0 A}{2(y_0 + y)}$ ,  $\Delta = N_c N_e - N_{ce} N_{ec}$ , and  $k = \frac{N_{ce}}{N_c} = \frac{N_{ec}}{N_e}$ . The equations presented in this section constitute a consistent mathematical model relating the input voltages applied to the actuator coils,  $v_{c1}$  and  $v_{c2}$ , to the position of the shaft,  $y$ .

### 3 Fuzzy Control

Conventional feedback control systems measure, relatively precisely, certain process variables, operate on these measurements with a control algorithm to produce precise command signals, and apply these command signals to the process to control its behavior in some desired way. The control algorithm generally relies on an explicit mathematical model of the system to be controlled and some expression of desired system performance. A



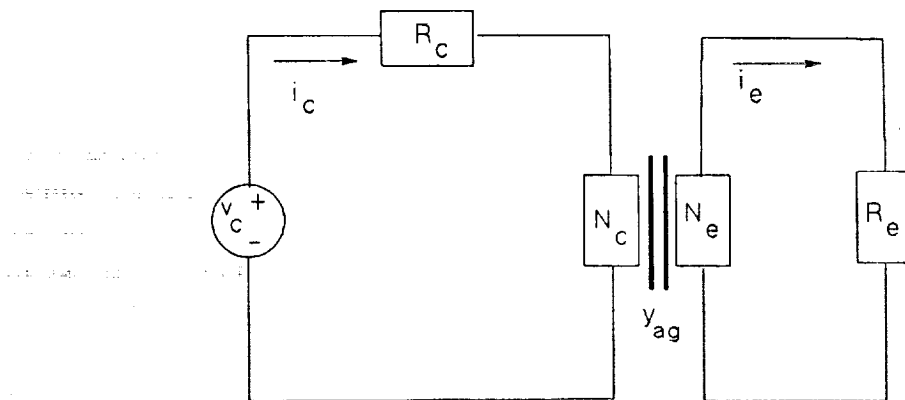


Figure 2: Circuit model of actuator.

crucial element in control algorithm design is the development of a suitable mathematical model of the system; in general, performance of the controlled system will be no better than the system model on which the control algorithm is based. The model should be neither too complicated, making the control algorithm too complex to implement, nor too simple, missing essential features of system behavior. Since most systems requiring automatic feedback control are dynamic and nonlinear, the development of a simple model that still captures the essence of important system performance characteristics is usually a time-consuming, and in some cases, impossible, task.

It is interesting to compare these automatic control systems with manual control systems where a human operator makes seemingly imprecise measurements, processes them rapidly in the brain, and produces the correct control command to, say, ride a bicycle. While it may not be impossible to build an automatic control system to control a bicycle (although we have never seen one), it would certainly be quite difficult. Yet, a young child can become a proficient rider after only a short training session with no knowledge whatsoever of the mathematics of bicycle dynamics. It is this paradox that led Zadeh [11] to the development of the theory of fuzzy sets, Mandami [12] to consider the linguistic synthesis of fuzzy control systems, and, most recently, Kosko [13] to explore its connections with neural networks in the adaptive control of dynamic systems.

As with neural network controllers, fuzzy controllers try to emulate the functions of the human brain. A fundamental difference between the two is that neural controllers assume no a priori knowledge of system behavior, while fuzzy controllers start with a linguistic description of whatever is known about the system. There is, however, a striking similarity at the implementation level between neural network controllers and adaptive fuzzy controllers [13].

### 3.1 Fuzzy Variables and Fuzzy Values

The notions of fuzzy control are rooted in the theory of fuzzy sets [11]. The basic difference between conventional (crisp) set theory and fuzzy set theory lies in the values assigned to the variables. Consider, for example, a variable called position,  $y$ . In crisp theory  $y$  could take on values, say, from 0m to +10m. At any particular point in time, the position of an object could be given by the value, say, 4m. In fuzzy theory, however, the values assigned to the position variable,  $y$ , are of not the familiar, crisp, numerical type but, rather, an unfamiliar, fuzzy, linguistic type; e.g. "close", or "far", or "very far". This is consistent with the child bicyclist's assessment of position relative to an upcoming tree. Since one of the strengths of fuzzy theory is that it is basically quantitative in nature, it remains to relate the fuzzy values "close", etc. to appropriate numerical values in a fuzzy way consistent with our notion of the meanings of the corresponding linguistic values. In the example considered above, "close", "far", and "very far" may be characterized by the distributions shown in Figure 3 where the abscissa is the distance from the tree and the ordinate is the degree to which "close", etc. is an accurate representation of the distance to the tree. Certainly, if the cyclist is about to hit the tree it is "close" while if it is 10m away it is not. If, however, it is 4m away it is only "close" to a degree; more specifically "close" is an accurate description of the distance 4m with degree 0.21, while "far" is an accurate description of this same distance with degree 0.64, and "very far" is not at all accurate and, so, is descriptive with degree 0.0. This subjective assessment of "closeness", etc. is introduced by the designer in the development of these distributions, or as they are known in fuzzy theory, "membership functions". To summarize, it is correct to think of the the fuzzy values "close", etc. as "fuzzy numbers" whose relationship to "crisp numbers" is provided by a defining membership function.

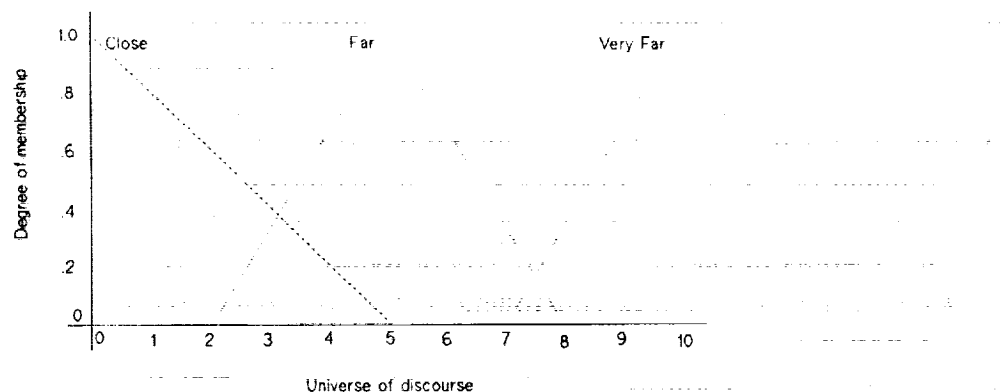


Figure 3: Membership functions for the fuzzy values "Close", "Far", and "Very Far" of the fuzzy variable "Position".

### 3.2 Fuzzy Functions

Analogous to the function of crisp mathematics that maps crisp input variables into crisp output variables, fuzzy mathematics uses a relational matrix to map fuzzy input variables into fuzzy output variables. The relational matrix is constructed from a linguistic rule base relating fuzzy input variables to fuzzy output variables. The linguistic rule base may be generated from a set of logical implications of the "IF-THEN" type. Consider, for example, a system with two fuzzy input position variables,  $x$  and  $y$ , and one fuzzy output steering variable  $\theta$ . Let the possible fuzzy values of  $x$  be "left" (L), "center" (C), and "right" (R), let the possible fuzzy values of  $y$  be "close" (C), "far" (F), and "very far" (VF), and the possible fuzzy values of  $\theta$  be "left" (L), "center" (C), and "right" (R). A brief linguistic rule base might then consist of the following logical implications:

1. IF [ $x$  is L and  $y$  is C] THEN [ $\theta$  should be R]
2. IF [ $x$  is R and  $y$  is C] THEN [ $\theta$  should be L]
3. IF [ $x$  is C and  $y$  is V] THEN [ $\theta$  should be C]

The relational matrix embodying these rules is shown in Figure 4, and is seen to be a concise display of the relationship between the pairs of fuzzy values of the fuzzy input variables and the fuzzy values of the fuzzy output variable. It is interesting to note that the relational matrix is not necessarily full. An important and powerful aspect of fuzzy control is that only those rules that are well known need be specified, the fuzzy calculations will "interpolate" or "extrapolate" to fill in missing rules. The fuzzy calculations will also resolve conflicting rules in an optimal way consistent with the specified linguistic rule base and defined fuzzy variables.

		Fuzzy position variable, $x$		
		L	C	R
Fuzzy position variable, $y$	V		C	
	F			
	C	R		L

Figure 4: Relational matrix mapping fuzzy input variables  $x$  and  $y$  to fuzzy output variable  $\theta$ .

### 3.3 Fuzzy Controller

Fuzzy control systems inevitably interact with the physical world of crisp measurements and actuators. On input to the controller, crisp values of crisp variables are converted to fuzzy values of fuzzy variables according to the membership function of the fuzzy variable. For example, in Figure 3 a crisp value of "4" of the crisp variable *position* would take on two fuzzy values "close" and "far" of the fuzzy position variable. The membership functions indicate that the crisp value "4" is the fuzzy value "close" with degree 0.21 and the fuzzy value "far" with degree 0.64. Thus, a single measurement of a crisp variable may activate a number of rules in linguistic rule base or, equivalently, the relational matrix. Each rule will operate on its fuzzy input variables, and their membership functions, to produce a modified membership function, or fuzzy value, for the the fuzzy output variable. The specific form of the output membership function may be determined either by the correlation-minimum or the correlation-product inferencing technique [13]. Since more than one rule may be activated by a single measurement it follows, then, that a number of fuzzy values of the output may also be generated. The output membership functions generated by the firing of several rules may be combined in a number of different ways to produce a single crisp output to activate a physical actuator. Two commonly used methods are the mean-of-maxima and the centroid methods[13].

The fuzzy controller under development for the magnetic bearing has two fuzzy input variables, position  $y$  and, change in position  $dy$ ; and one fuzzy output variable, actuator voltage  $v$ . Each fuzzy variable may take on each of seven fuzzy values: "negative large" (NL), "negative medium" (NM), "negative small" (NS), "zero" (ZE), "positive small" (PS), "positive medium" (PM), and "positive large" (PL). The fuzzy values of the input variables are shown over their corresponding universe of discourse in Figure 5. The universe of discourse ranges from -5 volts (corresponding to a shaft position of  $-19\mu\text{m}$ ) to +5 volts (corresponding to a position of  $+19\mu\text{m}$ ). Fuzzy values are trapezoidal in shape with a maximum overlap of 25%, and are narrower near zero to provide finer control close to the desired value.

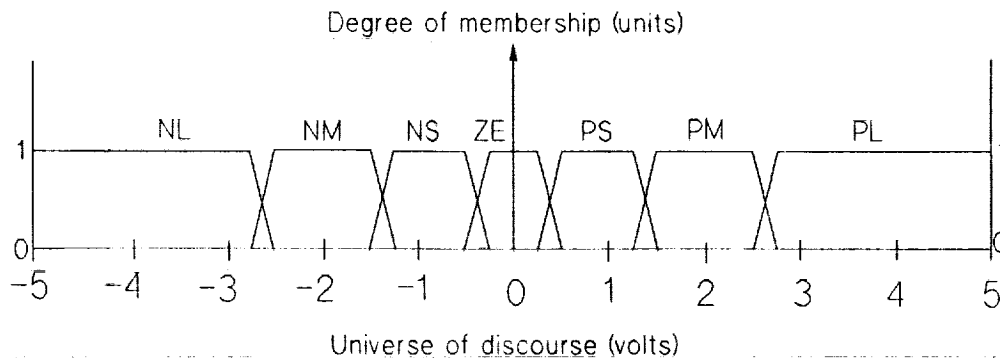


Figure 5: Fuzzy values of input variables  $y$  and  $dy$ .

Fuzzy values of the output variables are shown in Figure 6. They are triangular in

shape, have a maximum overlap of 25%, and are closer together near zero to provide finer control. The exact shapes and locations of the fuzzy input and output variables are design parameters whose optimal values are found by numerical experimentation.

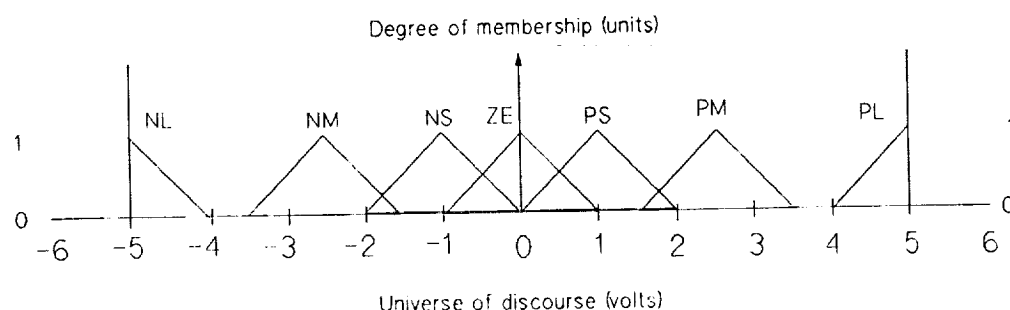


Figure 6: Fuzzy values of the output variable  $v$ .

The  $7 \times 7$  relational matrix relating the fuzzy input pairs to fuzzy values of the output is shown in Figure 7. The relationship between the relational matrix and the corresponding set of forty nine IF-THEN implications is obvious.

The correlation-minimum inference procedure is used to process activated rules resulting in a truncation of the output membership function at the minimum value of the two input membership functions. Note that since a maximum of two input values overlap, a maximum of four (as opposed to a possible maximum of forty nine) rules can be activated at once. Combining of output fuzzy values and subsequent defuzzification is performed using the centroid method.

## 4 Performance of Magnetic Bearing with Fuzzy Controller

A linearized version of the nonlinear model presented in Section 2 was programmed using Matlab to test the performance of the fuzzy controller. Figure 8 shows the response to a  $3.8\mu\text{m}$  (1 volt) step demand change in position. The figure shows that the fuzzy controller was successful in stabilizing the bearing and that response time is short. Sampling frequency was 10 K Hz. Oscillations are small and can be further reduced by reducing the size of the fuzzy sets representing zero error. Steady state error can be further reduced by adding an integral mode to the controller. These results are not surprising since the present fuzzy controller uses only position and velocity inputs and is essentially operating as a proportional-plus-derivative controller. Additional work is being conducted to correct these deficiencies. Several promising adaptive control policies are being investigated including modifying the input fuzzy set sizes and overlap, the output fuzzy set centroids, and the scaling gains  $K_e$ ,  $K_c$ , and  $K_v$ . Best results were obtained with 25% set overlap and  $K_e = 1$ ,  $K_c = 18$ , and  $K_v = 5$ .

		Change in position, $dy$						
		NL	NM	NS	ZE	PS	PM	PL
Position, $y$	NL	NL	NL	NM	NM	NS	NS	ZE
	NM	NL	NM	NM	NS	NS	ZE	PS
	NS	NM	NM	NS	NS	ZE	PS	PS
	ZE	NM	NS	NS	ZE	PS	PS	PM
	PS	NS	NS	ZE	PS	PS	PM	PM
	PM	NS	ZE	PS	PS	PM	PM	PL
	PL	ZE	PS	PS	PM	PM	PL	PL

Figure 7: Relational matrix for the magnetic bearing controller.

## 5 Architecture for a Fuzzy VLSI Chip

The architecture of a fuzzy VLSI chip is outlined in Figure 9. The basic fuzzy control algorithm is contained on a single chip. Rules are downloaded from a host computer at start-up and can be modified by the host computer later. The chip is of the all-digital type so off-chip A/D and D/A converters are required. The fuzzy control algorithm has four parts: 1) input calculations, 2) input membership determination, 3) rule evaluation, and 4) output defuzzification as described below.

### 5.1 Input Calculations

The single input to the chip is the position error in volts. The current error  $er(n)$  and the previous error  $er(n-1)$  are each stored in a separate registers. The current change in error  $ch(n) = er(n) - er(n-1)$  is computed and stored in a third register. The variables  $ER$  and  $CH$ , used in membership function determination are found by multiplying  $er(n)$  by  $K_e$  and  $ch(n)$  by  $K_c$ , respectively. The scaling gains  $K_e$  and  $K_c$  are downloaded from the host computer and may be modified as required.

### 5.2 Input Membership Determination

The input membership determination is made by a table look-up. There are two look-up tables one for  $ER$  and one for  $CH$ . The output of the table look-up is the modified fit vector  $(A, m_A, m_B)$ . Each look up table is of size 3 by  $m$  by  $n$ .

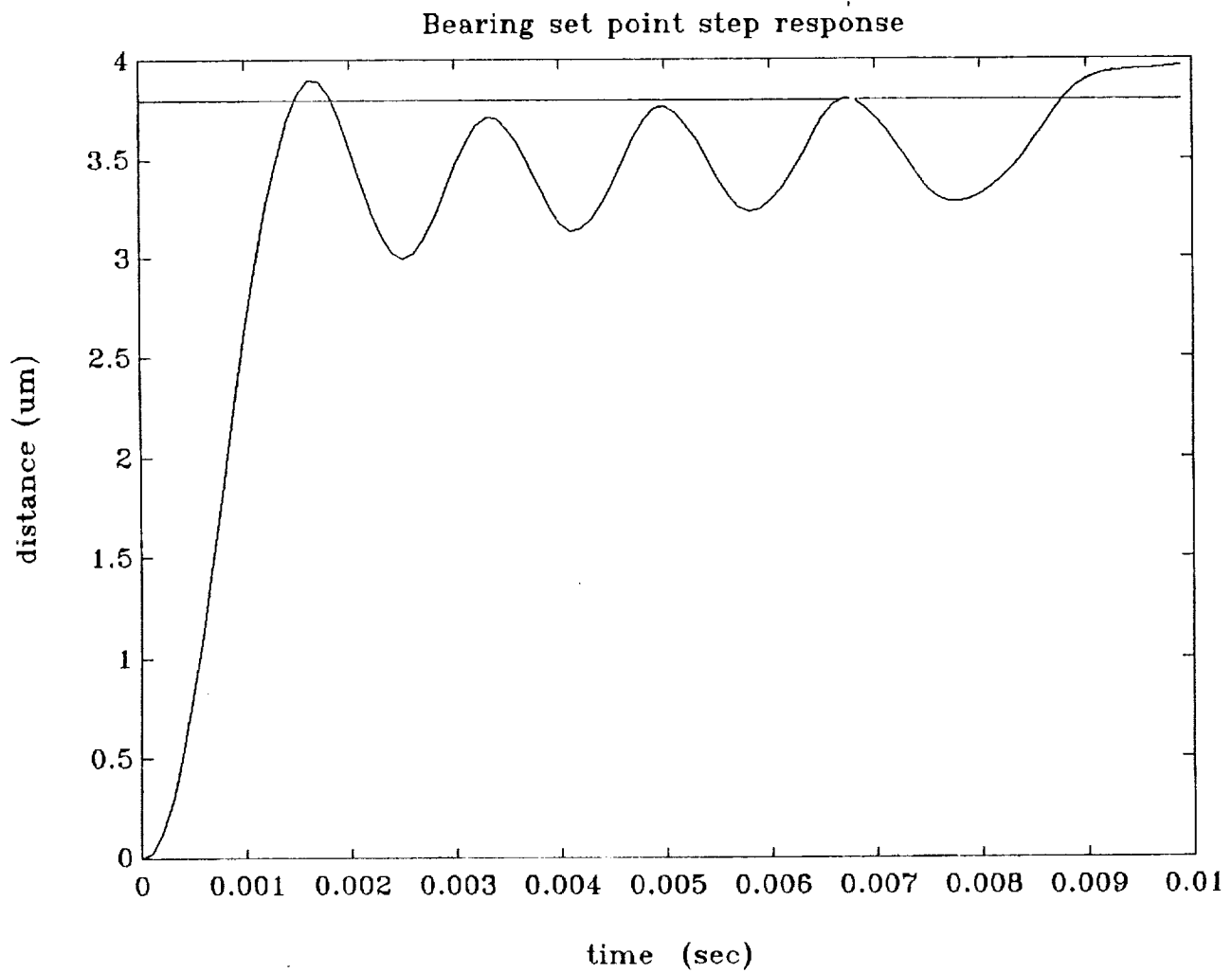


Figure 8: Response of fuzzily controlled magnetic bearing to  $3.9\mu\text{m}$  step change in position.

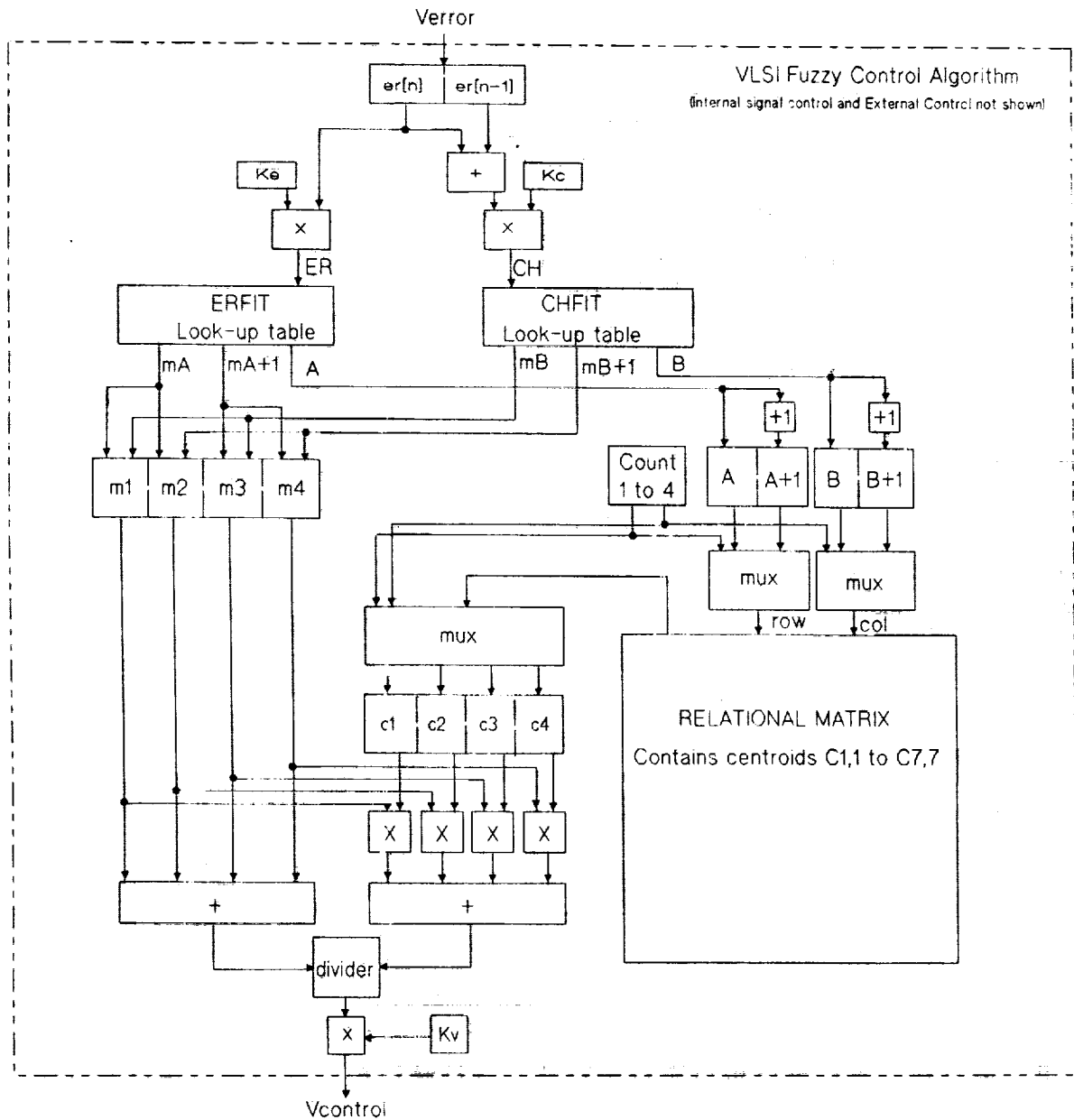


Figure 9: Architecture of a fuzzy VLSI chip.



### 5.3 Rule Evaluation

Four rules are evaluated for each input pair. Each evaluation finds the minimum of the input fuzzy sets and the centroid of the output fuzzy set. A 4 by  $n$  hold the minimum input membership values and a 4 by  $m$  register hold the corresponding centroids.

### 5.4 Output Defuzzification

Defuzzification is done in three steps. First, the minimum membership value is multiplied by the centroid for each of the rules activated. Second, each of these products is summed to produce; at the same time each of the minimum membership values is also summed. Finally, the sum of the minimum membership-centroid products is divided by the sum of the minimum memberships to produce the desired result. The result is then multiplied by an output voltage scaling gain  $K_v$ .

## 6 Summary and Conclusions

A mathematical model of a magnetic bearing was presented and was used to develop a computer simulation model to test alternative magnetic bearing control systems. A fuzzy control system for was developed and tested by computer simulation. Initial results show that the fuzzy controller stabilizes the magnetic bearing and produces acceptable steady-state and transient behavior. Further research is being conducted to optimize the fuzzy controller and to develop suitable adaptive algorithms. Particular emphasis is being placed on achieving zero steady-state error and rejecting acceleration disturbances. Performance comparisons between the fuzzy controller and a linear-quadratic-gaussian regulator are being conducted. A candidate VLSI chip architecture has been proposed to implement the fuzzy control algorithm and provide rapid sampling for real-time control. VLSI-based fuzzy control appears feasible for real-time control of uncertain nonlinear systems like an active magnetic bearing.

## 7 Acknowledgements

This work was supported in part by a grant from the NASA Space Engineering Research Center at the University of Idaho.

## References

- [1] Forbes Magazine, May 1, 1989.
- [2] C. Keung et. al., Design and Fabrication of a Long-Life Stirling Cycle Cooler for Space Application, *Philips Laboratories*, November, 1990.

- [3] R.D. Williams, F.J. Keith, and P.E. Allaire, Digital Control of Active Magnetic Bearings, *IEEE Transactions on Industrial Electronics*, February, 1990.
- [4] H.N. Chen and M.S. Darlow, Magnetic Bearing with Rotating Force Control, *ASME Journal of Tribology*, January, 1988.
- [5] F.J. Keith, R.D. Williams, and P.E. Allaire, Digital Control of Magnetic Bearings Supporting a Multi-Mass Flexible Rotor, *Tribology Transactions*, 1990, Vol 33,3,307-314.
- [6] H.M. Chen, Magnetic Bearings and Flexible Rotor Dynamics, *Tribology Transactions*, 1989, Vol 32, 1, 9-15.
- [7] R.R. Humphris, et. al. Effect of Control Algorithms on Magnetic Journal Bearing Properties, *ASME Journal of Engineering for Gas Turbines and Power*, October, 1986.
- [8] M.B. Scudiere, R.A. Willems, and G.T. Gillies, Digital Controller for a Magnetic Suspension System, *Review of Scientific Instrumentation*, August, 1986.
- [9] J.J. Feeley, A. Law, and F. Lind, Digital Control of a Magnetic Bearing in a Cryogenic Cooler, *NASA SERC Symposium on VLSI Design*, 1990.
- [10] J.J. Feeley, D.J. Ahlstrom, and G.M. Niederauer, Optimal Control of a Magnetic Bearing, to be submitted for publication, 1991.
- [11] L. Zadeh, Fuzzy Sets, *Information and Control*, 1965, Vol 8, pp. 338-353.
- [12] E. Mandami and S. Assilian, An Experiment in Linguistic Synthesis with a Fuzzy Controller, *International Journal of Man-Machine Studies*, 1975, Vol 7, pp. 1-13.
- [13] B. Kosco, *Neural Networks and Fuzzy Systems*, Prentice-Hall, Inc., 1992.

# Direct Kinematics Solution Architectures for Industrial Robot Manipulators: Bit-Serial Versus Parallel

J. Lee

Department of Electrical Engineering  
University of Houston  
Houston, TX 77204-4793

K. Kim

Superconducting Super Collider Lab.  
2550 Beckleymeade Avenue  
Dallas, TX 75237

**Abstract** - This paper investigate a VLSI architecture for robot direct kinematic computation suitable for industrial robot manipulators The Denavit-Hartenberg transformations are reviewed to exploit a proper processing element, namely an augmented CORDIC. Specifically, two distinct implementations are elaborated on, such as the bit-serial and parallel. Performance of each scheme is analyzed with respect to the time to compute one location of the end-effector of a 6-links manipulator, and the number of transistors required.

## 1 CORDIC Techniques

The matrix  $A_j$  describing the  $j$ th link is proposed to be implemented via 4 CORDICs: parallel two for the  $w$ -axis operation, and another parallel two for the  $x$ -axis. Since, the rotation and translation are disjoint each other, the 4 CORDIC can be done via a 2-stages cascade [5].

Let the  $j$ th joint orientation vector denote by  $p_j$ , where  $p_j = A_j p_{j-1}$ . Consider an intermediate vector  $p_j^A$ , between  $p_j$  and  $p_{j-1}$ :

$$p_j = \text{Trans}(w_{j-1}, d_j) \text{Rot}(w_{j-1}, \theta_j) p_j^A : \text{stage} - 1 \quad (1)$$

$$p_j^A = \text{Trans}(x_j, a_j) \text{Rot}(x_j, \psi_j) p_{j-1} : \text{stage} - 2. \quad (2)$$

One set of transformations for each stage, i.e.  $\text{Trans}(w, d) \text{Rot}(w, \theta)$ , is a block-diagonal matrix and can be orthogonally implementable by two 2x2 matrix transformations. Note that is implementable through an augmented PE, rather two different PEs, observing that  $\text{Trans}(w, d)$  is a trivial operation. Then,

$$p_j = \begin{bmatrix} x_j \\ y_j \\ \cdot \\ w_j \\ 1 \end{bmatrix} = \begin{vmatrix} \text{Rot}(w_j, \theta_j) & : & O \\ \cdot & \cdot & \cdot \\ O & : & \text{Trans}(w_j, d_j) \end{vmatrix} p_j^A. \quad (3)$$

$p_j$  (also, similarly for  $p_j^A$ ) is decomposed into two blocks, e.g the first two elements of  $p_j$  becomes one vector  $X_j$ :

$$p_j = [X_j; w_j, 1]^t = [Rot(w_j, \theta_j); w_j + d_j, 1], \quad (4)$$

where  $w_j$  is for the w-axis component of the vector  $p_j$ , and  $X_j$  for x- and y-axis components rotated by  $\theta_j$ . In a similar way, for  $p_j^A$  we can choose a rotated vector of y- and w-axis disjointly through axis shuffling. Finally, consecutive n-pairs of rotation and translation can be implemented via a 2n-stages cascade. We will name each stage as a macro-PE (or, an augmented PE), which can be 2n-pipelined to compose an n-links computation processor. Not to differentiate the two different sets of transformations, w-axis and x-axis respectively, we employ index  $i$  in unified notations for a macro-PE: for a reference axis  $w_i$ , there are rotation of  $\theta_i$  and translation of  $d_i$ ,  $\bar{X}_{i-1} = (x_{i-1}, y_{i-1})$  for an input, and  $\bar{X}_i = (x_i, y_i)$  for an output.

Each macro-PE including one  $Trans(w_i, d_i)$  and one  $Rot(w_i, \theta_i)$  can be implemented, as in Figure 1.a. One-joint processor is shown in Figure 1.b. Finally, for a 6-joints system, Figure 1.c shows a fully pipelined structure.

From this point, we will concentrate on implementation of a macro-PE. Observing that  $Rot$  and  $Trans$  functions are disjoint each other, let us isolate the rotation part at first. This vector rotation for  $\bar{X}_i = (x_i, y_i)$  by the angle  $\theta_i$  can be realized by an iteration algorithm called CORDIC [4] instead of computing trigonometric functions and applying matrix multiplication. CORDIC realizes a vector rotation by a partial sum of micro-angle rotations with a pre-fixed sequence of angles. When the rotation macro-angle is represented as a sum of decomposed micro-angles, i.e  $\theta_i = \sum_{k=0}^n \bar{\theta}_{i,k}$ ,

$$\bar{X}_i = \prod_{k=0}^n k_k \begin{bmatrix} 1 & -\tan\theta_{i,k} \\ \tan\theta_{i,k} & 1 \end{bmatrix} \bar{X}_{i-1} \quad (5)$$

where  $k_k = \cos\theta_{i,k}$  is a micro-scale composing a final scale factor, explained later. Such a specific form of the pre-fixed micro-angle sequence as  $\tan^{-1} 2^{-i}$ , is attractive for VLSI implementation since it is composed only of additions, shiftings, and a arctangent lookup table. For the simplicity of notation, subscript  $i$  indexing a certain stage will be omitted, and  $X, Y$  and  $Z$  stand for abridged notations for those having subscript  $i$ .

**Non-redundant :** The micro-iterations of the conventional (hereafter, it will be called non-redundant ) CORDIC are 3 linear recursive equations: X recurrence (X-rec.), Y-recurrence (Y-rec.) and Z-recurrence (Z-rec.) [4].

$$\begin{aligned} X[i+1] &= X[i] + \sigma_i 2^{-i} Y[i] \\ Y[i+1] &= Y[i] - \sigma_i 2^{-i} X[i] \\ Z[i+1] &= Z[i] - \sigma_i \tan^{-1} 2^{-i} \end{aligned} \quad (6)$$

With an initial value of  $Z[0] = \theta_i$ , CORDIC rotates initial values of  $X[0]$  and  $Y[0]$ , to the last value  $X[n]$  and  $Y[n]$ , while making  $Z[i]$  close to zero, so that  $Z[n]$  is forced to be zero. With  $n$  number of iterations,  $n$ -bit accuracy of  $X$  and  $Y$  in the output can be achieved.

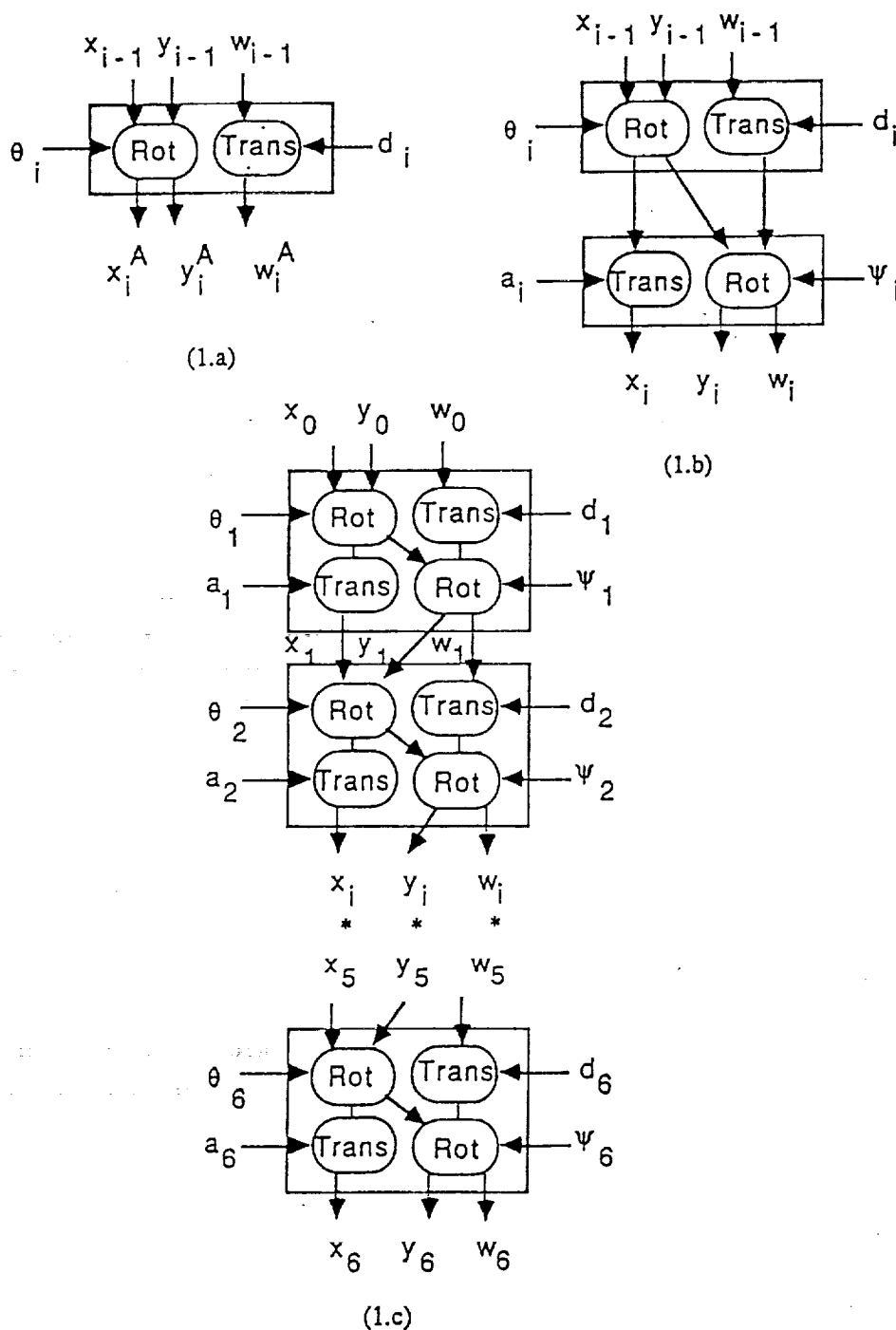


Figure 1: CORDIC-based Pipelined Architecture for Direct Kinematics Computation: a. A macro-PE, One-stage from an orientation to an intermediate, b. 2-stages cascade, An  $A_i$  transformation module for a link, c. A complete pipelined Computation Module for 6-links system.

For a known angle, the direction of the rotation,  $\sigma_i$  can be pre-computed or calculated one by one on-the-fly using the following selection function.

$$\sigma_i = \begin{cases} 1 & \text{if } Z[i] \geq 0 \\ -1 & \text{if } Z[i] < 0 \end{cases} \quad (7)$$

The CORDIC rotation does not preserve the input norm. To get a rotated vector having the same length as the input  $(X[0], Y[0])$ ,  $X[n](Y[n])$  needs to be compensated by a scaling factor  $K$

$$K = \frac{\| [X[n], Y[n]]^t \|}{\| [X[0], Y[0]]^t \|} = \prod_{i=0}^{n-1} \sqrt{1 + \sigma_i^2 2^{-2i}}, \quad (8)$$

where  $\|\cdot\|$  stands for the norm of the vector. Note that  $K$  is *constant* for the non-redundant scheme since  $\sigma_i$  is in  $\{-1, 1\}$ .

**Redundant :** Non-redundant CORDIC is slow inherently with delay of  $O(n^2)$  due to its recursiveness and serial dependency, since a micro-rotation with delay  $O(n)$  should be finished before processing the next micro-rotation. Delay performance of a macro-rotation ( $n$  micro-rotations) can be improved from  $O(n^2)$  to  $O(n)$  by using redundant arithmetic (carry-free addition such as carry save or signed-digit addition) to determine the direction of the rotation  $\hat{\sigma}_i$ , based on an estimate instead of an exact value [9]. The redundant arithmetic gives a delay of  $O(1)$  instead of  $O(n)$ , and the estimation of direction is necessary not to erode the advantage of  $O(1)$ . This requires the modification of the recurrences and selection function. This redundant CORDIC scheme produces the output about 4 times faster than the non-redundant. However, it introduces additional cost since the scale factor  $K$  is variable depending on a macro-angle by allowing  $\hat{\sigma}_i$  to be in  $\{-1, 0, 1\}$ .

**Constant-Factor-Redundant :** To reduce implementation cost of redundant CORDIC, it would be good to have a constant scale factor by forcing  $\hat{\sigma}_i$  in  $\{-1, 1\}$ . However, since  $\hat{\sigma}_i$  is determined from an estimate, there arises a convergence assurance question. There was proposed a scheme appending correcting iteration stages at proper positions [10]. Along to this idea, the number of extra correcting iterations is further reduced by dividing the micro-iterations (for  $i = 0$  to  $i = n - 1$ ) into two groups: one group where the direction of the rotation is in  $\{-1, 1\}$  for  $i = 0$  to  $i = n/2$  and the other in  $\{-1, 0, 1\}$  for  $i = (n + 1)/2$  to  $i = n - 1$  correcting iterations by 50 % since correcting iteration is not needed for the second half of the micro-iterations and we still obtain a constant scale factor  $K$  since the value of  $K$  in  $n$ -bit precision does not depend on the  $\hat{\sigma}$  value for  $(n + 1)/2 \leq i \leq (n - 1)$ .  $Z$ -recurrence also can be modified so that  $\hat{\sigma}_i$  is determined quickly by looking at a few most significant bits. This new scheme is called Constant-Factor-Redundant-CORDIC(CFR-CORDIC). The modified recurrences and selection functions for the scheme are described below.

$$\begin{aligned} X[i + 1] &= X[i] + \hat{\sigma}_i 2^{-i} Y[i] \\ Y[i + 1] &= Y[i] - \hat{\sigma}_i 2^{-i} X[i] \\ U[i + 1] &= 2(U[i] - \hat{\sigma}_i 2^i \tan^{-1} 2^{-i}) \end{aligned} \quad (9)$$

where  $U[i]$  is for the implementation simplicity, which is equal to  $2^i Z[i]$ , and the selection function is given as follows:

$$\hat{\sigma}_i = \begin{cases} 1 & \text{if } \hat{U}[i] > 0 \\ & \text{or } \hat{U}[i] = 0 \cap i < n/2 \\ 0 & \hat{U}[i] = 0 \cap i \geq n/2 \\ -1 & \text{if } \hat{U}[i] < 0 \end{cases} \quad (10)$$

When  $t$  fractional bits are used in the estimate value, i.e.,  $\hat{U}[i]$  is computed using  $t$  fractional bits of redundant representation of  $U[i]$ , the following correcting iteration need to be included, where the interval between indexes of correcting iterations should be less than or equal to  $(t - 1)$  up to the last iteration index equal to  $n/2$ . When the correction stage is necessary at the  $j$ th step of micro-iteration,

$$U^C[j + 1] = U[j + 1] - 2\hat{\sigma}_j^C 2^j \tan^{-j} 2^{-j} \quad (11)$$

with the direction of the rotation  $\hat{\sigma}_j^C$  determined from the same selection function of eq.( 10), except being decided based on  $\hat{U}[j + 1]$  instead of  $\hat{U}[i]$ .

So far, we discussed about recursive structures of several CORDIC schemes to implement the rotation part in the basic PE, as depicted in Figure 1. The PE, augmented by a translator, necessitates scaling operation at each stage, because shuffling of the output at each stage makes continuous accumulation of the scaling factor complex to be processed at the final stage. The scaling operation has been solved either by an explicit way or an implicit. The explicit way is dividing the rotated vector by a constant, which is known for the non-redundant, to be calculated while running the micro-steps of CORDIC [4,9]. The division can be processed by another CORDIC (in a linear mode) or a divider. The implicit approach reconfigures the sequence of micro-iterations of the CORDIC, eventually to have a different norm from that without scaling micro-iterations. Scaling micro-iterations target in general at making the adjusted scaling factor in a form of  $2^i$  or 1, which can be easily set to the unit size. Each micro-iteration can be composed of i) reduction axis-scaling [11], ii) repetition of vector-scaling, iii) expansion axis-scaling or combinations thereof [12]. Relevant issues regarding solution search are to be further studied, more than the greedy method or the decomposed [13]. In summary, the explicit scaling almost doubles the system complexity, while the implicit increases 25 % for the non-redundant and about 30 % for the redundant.

## 2 Application to Direct Kinematics

In this section, we design an architecture for the direct kinematics computation, based on CFR-CORDIC. The data-path is the parallel. To analyze its performance, we will define a new measure, namely one-position calculation time. Via this measure, we will also analyze performance the bit serial architecture similarly implementable as in

## 2.1 Performance Measure

Let's define the following parameters.

$b_i$  : the number of bits in each input  $x, y$  and  $w$

$b_f$  : the number of bits in each output

$n_f$  : the number of links (=6)

$f_c$  : the available data shift rate

$\Delta$  : the step time per micro CORDIC iteration

$f_i$  : the input bit rate

Additionally, we define a measure parameter  $T_\Delta$ ,

$$T_\Delta = \text{step-time}(\Delta) * \text{number of steps},$$

to compare the performance of various schemes. For a discrete element implementation,  $\Delta$  corresponds to one single external clock time  $1/f_c$ . Note that  $\Delta$  varies depending on a particular implementation of a macro-PE. Without loss of generality, let's define the unit of  $\Delta$  to be 1 for one-bit full addition time. The input processing rate can be alternatively interpreted as

$$\frac{f_i}{b_i} < \frac{1}{T_{\Delta_1}}, \quad (12)$$

which limits the maximum rate of input vector sampling to be processable through an implemented processor.

## 2.2 Performance Comparison

**Bit Serial:** A macro-PE using serial data path and arithmetic units for CORDIC is shown in Figure 2 [6]. Figure 2.a shows symmetric components of a bit-serial PE in  $x, y$  and  $w$  representation, and Figure 2.b is for the detail of each block (X-recurrence or Y-recurrence) employing bit serial arithmetic. W-recurrence is in Figure 2.c, and Z-recurrence in Figure 2.d. The  $x$  and  $y$  components of the input vector  $X_{i-1}$  are taken initially as  $X[0]$  and  $Y[0]$ , and the initial angle  $Z[0]$  is set to the corresponding joint angle. After performing  $n$  micro-iterations, CORDIC produces  $n$ -bit precision outputs leading to  $X_i$ .

In the serial scheme without macro-pipelining, denote a basic step-time as  $\Delta_1$ , which is equivalent to  $\Delta$ . To use one adder recursively  $n_f$  times to process an  $n_f$  links,

$$T_{\Delta_1} = \Delta_1 * n_f(b_f + b_i(b_i + \log_2 b_i)),$$

where the output has  $b_f$  bits buffer.

**CFR-Redundant Parallel :** To increase the throughput of the previous, the bit-serial PEs can be substituted by those using parallel arithmetic. When parallel arithmetic and non-redundant CORDIC are adopted, the corresponding parameter becomes

$$T_{\Delta_2} = \Delta_2 * n_f(b_i + \log_2 b_i)$$

where  $\Delta_2$  equals to the time for one micro-rotation (time for variable shifter plus time for carry-propagate addition), approximately  $2 \log_2 b_i$  assuming fast variable shifter and carry-propagate adder. The step time can be further shortened by adopting CFR-CORDIC,



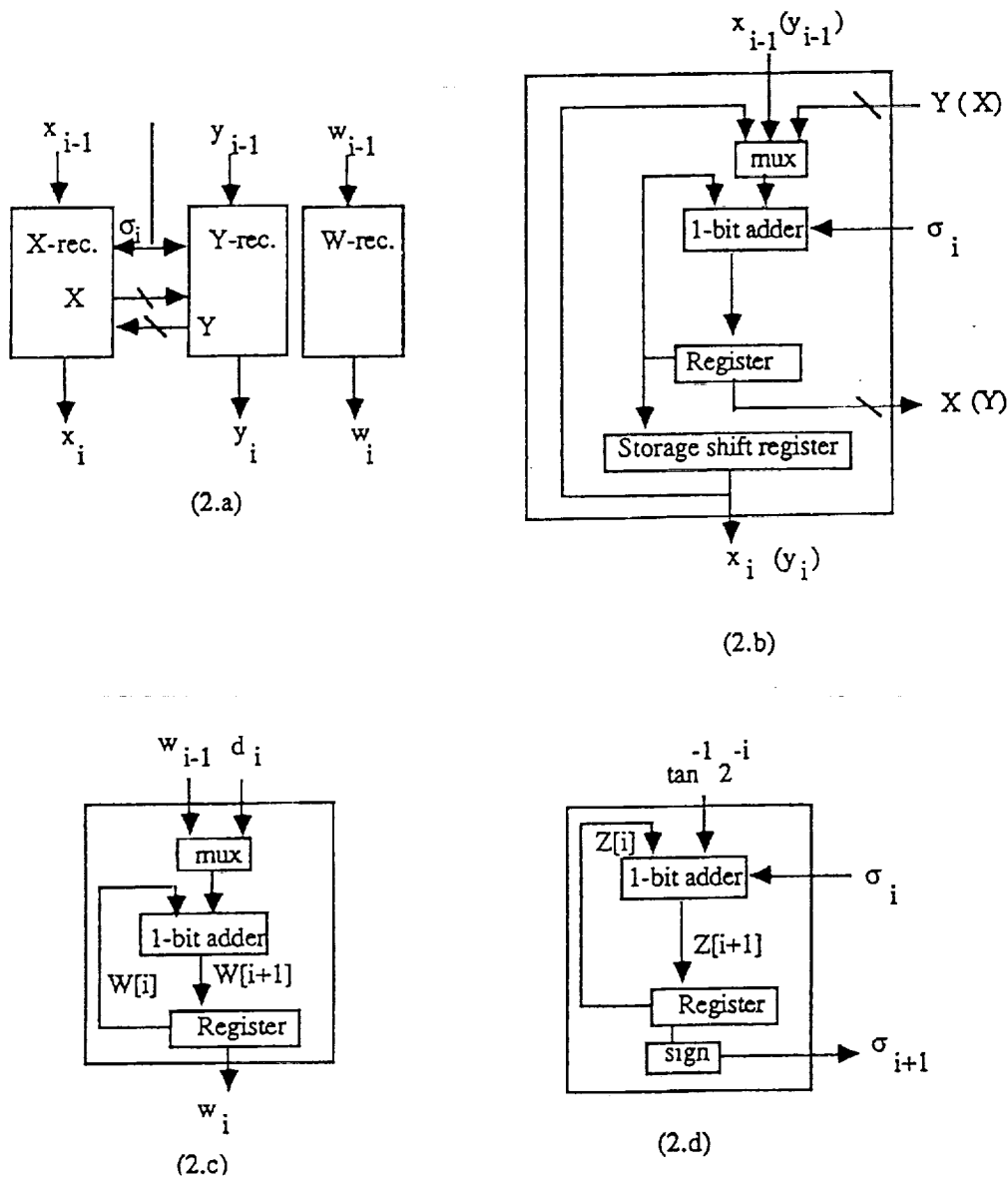


Figure 2: A bit-serial PE : a. A macro-PE with X-, Y- and W-recurrence, b. Detail of either block, c. W-recurrence, d. Z-recurrence.

where a carry-free adder (signed-digit adder) is replaced for carry-propagate adder. Figure 3.a shows a macro-PE in components, and Figure 3.b is for the detail of each block (X-recurrence or Y-recurrence) employing parallel/redundant arithmetic. Z-recurrence is in Figure 3.c.

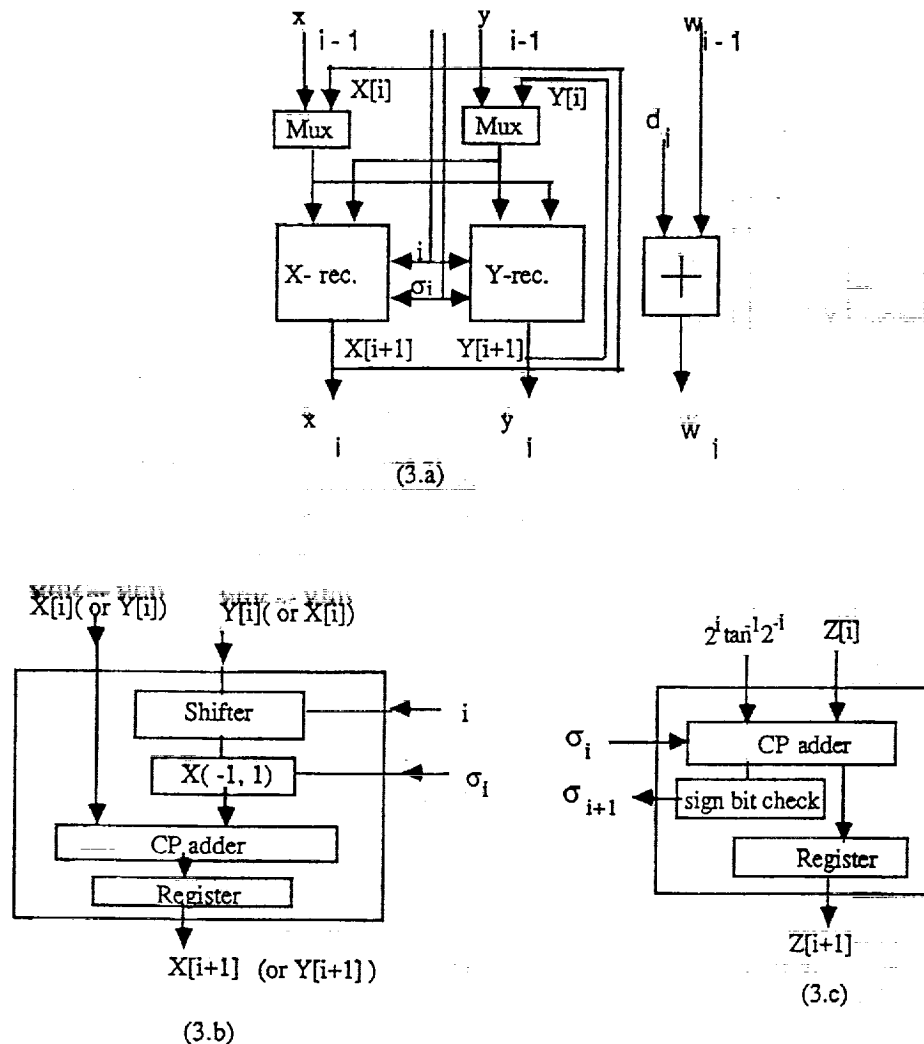


Figure 3: A parallel/redundant PE: a. A macro-PE with X- and Y-recurrence, b. Detail of either block, c. Z-recurrence.

Description	$\Delta_i/\Delta$	$T_{\Delta_i}$	Processing rate	TRs estimate
Bit-serial (parallel)	1	$1200\Delta$	600K 4M	2K 12K
Parallel(CFR) (parallel)	5	$500\Delta$	2M 10M	6K 40K

Table 1: Time and complexity comparison

In this case, the sign of  $Z[i]$  at the  $i$ th micro-iteration can not be detected by looking at the most significant bit since  $Z[i]$  is in redundant number representation. To determine the sign of  $Z[i]$  quickly by looking at a few significant bits, CFR-CORDIC uses an estimate of shifted- $Z[i]$  ( $U[i]$ ) using  $t$  fractional bits. As discussed earlier, the number of fractional bits used for the estimate also determines the frequency rate of a correcting iteration: more fractional bits are used, less number of correcting iterations are required. Let the number of correcting iterations denoted by  $\eta$ . The corresponding  $T_{\Delta}$  becomes

$$T_{\Delta_3} = \Delta_3 * n_f(b_i + \log_2 b_i + \eta)$$

where  $\Delta_3$  equals to the time for carry-free addition plus the time for the maximum of a selection function and a variable shifter, approximately  $(1 + \log_2 b_i)$ . Note that a practical number of correcting iterations is much smaller than  $b_i$ , e.g. 1 for the 16bit resolution. Hence, we can approximate  $T_{\Delta}$  to be that for the redundant without a correcting iteration.

For a case,  $b_i = 12$ ,  $b_f = 16$ , the estimated  $T_{\Delta}$  is summarized in Table 1. To get first order estimates of available speed and area, we use a figure that one full adder (also one bit shifter) requires approximately 50 TRs and one 20nsec clock cycle [14].

### 3 Conclusion

We have examined various kind of CORDIC schemes as a macro-PE module for the direct kinematics processor, and showed that its micro-level regularity is suitable for VLSI implementation, depicted along with specific schematics which include the conventional non-redundant, the redundant and the Constant-Factor-Redundant schemes. The cost-effectiveness of selected architectures has been analyzed using bit-serial, parallel or pipelined structure with respect to the time and the number of modules required, to compute one location of the end-effector for a 6-links manipulator, given a set of angle measurements. The comparison table exhibits the CORDIC-based robotics processor as a prospective solution in VLSI to be used for a wide range of kinematics calculation requirement, compromising the size versus speed.

## References

- [1] J. Denavit and R. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, pp.215-221, 1955.
- [2] P. Nanua, K. Waldron and V. Murthy, "Direct Kinematic Solution of a Stewart Platform," *IEEE Trans on Robotics and Automation*, Vol 6, No 4, pp.438-444, Aug. 1990.
- [3] D. Moldovan and G. Lee, "On the Use of Parallel Architectures for Robotic Manipulators: The Kinematics Problem," *Int. J. Robotics and Automation*, Vol 1, No 2, pp.47-53, 1986.
- [4] J. Walther, "A Unified Algorithm for Elementary Functions," *AFIPS Spring Joint Computer Conference*, pp.379-385, 1971.
- [5] C. Lee, "CORDIC-based Architectures for Robot Direct Kinematics and Jacobian Computation," *3rd Int. Symp. Intelligent Control*, pp.609-614, 1988.
- [6] R. Harber et. al., "Bit-serial CORDIC Circuits for Use in a VLSI Silicon Compiler," *Int. Conf. Circuit and System*, pp.154-157, 1989.
- [7] M. Kameyama, T. Matsumoto and H. Hideki, "Implementation of a High Performance LSI for Inverse Kinematics Computation," *IEEE Int. Conf. Robotics and Automation*, pp.757-762, 1989.
- [8] H. Kung, "Let's Design Algorithms for VLSI systems," *Caltech Conf. VLSI*, pp.65-90. 1979.
- [9] M. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix triangularization and SVD," *IEEE Trans. on Computers*, Vol. C-39, No 6, pp.725-740, June 1990.
- [10] N. Takagi, T. Asada and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation", Submitted to *IEEE Trans. on Computers*, 1989.
- [11] G. Haviland and A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. on Computers*, Vol C-29, No 2, pp.68-79, Feb. 1980.
- [12] J. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces," *Proc. of ICASSP*, pp.927-930, 1983.
- [13] J. Lee and T. Lang, "Matrix triangularization by fixed-point redundant CORDIC with a constant scale factor," *Proc. SPIE Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations*, July 1990.
- [14] J. Harding, T. Lang and J. Lee, "A Comparison of Redundant CORDIC Rotation Engines," *Int. Conf. Computer Design 91*, Oct. 1991.

# Simplified Microprocessor Design for VLSI Control Applications

K. Cameron

NASA Space Engineering Research Center for VLSI System Design  
University of Idaho, Moscow, Idaho 83843  
Phone: 208-885-6500 Fax: 208-885-7579

**Abstract-** A design technique for microprocessors combining the simplicity of RISCs with the richer instruction sets of CISCs is presented. They utilize the pipelined instruction decode and datapaths common to RISCs. Instruction invariant data processing sequences which transparently support complex addressing modes permit the formulation of simple control circuitry. Compact implementations are possible since neither complicated controllers nor large register sets are required.

## 1 Introduction

The design of microprocessors has evolved considerably since the introduction of the first microprocessor in 1971 [3]. Traditional microprocessors are extremely complicated machines that support hundreds of instructions and a dozen or so addressing modes. The dominance of such complex instruction set machines (CISC) has recently been challenged by much simpler processors which support only the most commonly used instructions. These processors are known as reduced instruction set computers (RISC) [8]. Traditionally, RISC processors:

- Support a small (reduced) instruction set of simple instructions which represent the most commonly used operations,
- Process instructions at the rate of one instruction per system clock cycle,
- Have a large on board register file for instruction or data cache.

The SPARC architecture is a scalable family of traditional RISC processors which was developed commercially. The architecture is deeply pipelined and depends heavily upon the compiler to efficiently map the register set and avoid forbidden instruction sequences [1]. Recently, the meaning of the term RISC processor has become quite blurred [6]. The IBM System/6000 purports to be a RISC implementation, but supports 184 instructions [5], which is a considerably larger instruction set than that of the 68020 microprocessor [7], which is generally considered to be a CISC machine.

The design methodology described here is targeted for applications which must be implemented in a small amount of circuitry (i.e. as a cell on a larger integrated circuit), while retaining medium to high levels of performance. The approach taken is to implement

a system which adheres to most, but not all, of the design concepts of a traditional RISC machine. Key points of the design approach investigated are listed below. They will each be described in more detail later.

- The processor supports a very small (reduced) instruction set. Only vital or frequently used operations are supported directly.
- The instruction set is orthogonally partitioned. As nearly as possible, bit fields in instructions mean the same things for all instructions. All addressing modes are supported in the same manner for all instructions.
- All instructions are processed using invariant execution sequences. This means that information flows through the datapath in precisely the same manner for all instructions.
- Both the datapath and the associated controller are deeply pipelined. The use of invariant execution sequences permits the construction of very deep, yet simple processing pipelines.
- Only a small internal register set is supported. The processor registers are memory mapped, allowing them to be accessed and updated with general memory reference instructions.
- The support of relatively complex addressing modes is important if the internal register set is small. Implemented consistently across the entire instruction set, they add little to the overall complexity of the machine.

Though a specific processor was implemented, the design methodology followed may be used to implement a large number of different RISC-like processors, each with different size-performance trade-offs.

## 2 Execution Cycle Pipeline

The data flow strategy of the microprocessor is the first item which must be designed. This includes data flow to and from memory as well as through the datapath of the processor itself. The performance required of the processor drives the choices made at this point. Different cost-performance ratios can be achieved through the use of different data flow strategies. A few of the possible tradeoffs are listed below:

- Processor word size?
- Separate address and data busses used to access memory?
- Separate instruction fetch and program data stores?
- Separate address generation and data processing units?
- Multiple data processing units?
- Pipeline depth?
- Data/instruction cache?

Ram Access	Fetch1	Addr0	Fetch2	Addr1	Fetch3	Addr2	Fetch4	Addr3
Decode		Inst1		Inst2		Inst3		Inst4
Alu		Addr1	Op0	Addr2	Op1	Addr3	Op2	Addr4
Adder			Addr1	Op1	Addr2	Op2	Addr3	Op3

Figure 1:  $\mu$ P Execution Sequence

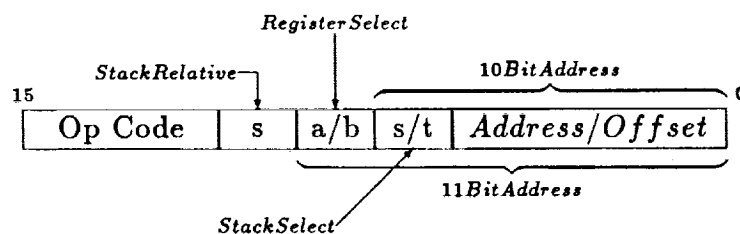
- Number of internal data/address registers?
- Maximum number of instructions?

Since the design implemented was to have moderate performance yet be compact, it was decided to build a processor with a 16 bit word, separate address and memory busses, shared data and instruction stores, combined address generation and data processing units, a deep pipeline, no cache, and a small set (4) of general registers. It was further decided that processor registers would be memory mapped, so separate instructions would not have to be provided to access either the processor registers or the registers associated with the accompanying IO subsystem.

Though shallow pipelines such as was used with RISC II [11] are relatively simple to design, it was decided from a performance stand-point that the machine should be deeply pipelined. A deep pipeline permits the construction of a high-throughput processor, since each stage of the pipe can operate independently on different portions of the problem at the same time. Deep pipelines, however, have the undesirable characteristic that any irregularity in the processing sequence for different instructions can lead to either the need for extremely complicated locking circuitry [5,1] or else the definition of a large number of forbidden instruction sequences to prevent data collisions. It was, therefore, decided that all instructions should share the same (though perhaps, a truncated) processing sequence. The memory execution sequence finally decided upon is shown in Figure 1. Several key points of this processing sequence are:

- Each RAM access is pipelined two clock cycles deep. This greatly eases all timing paths associated with RAM accesses.
- Data associated with an instruction "wraps" through the ALU/Adder twice. Once to calculate the associated address and once to process data. This strategy keeps the datapath completely utilized at all times.
- Data processed during one instruction is available for subsequent processing on the very next instruction.
- One instruction is executed every two clock cycles.

Mode	Invoked	Description
Direct	$s = 0$	Effective Address is part of instruction.
Indexed	$s = 1$ <i>Offset</i> $\neq 0$	Effective Address is contents of referenced stack pointer plus signed offset.
Stack	$s = 1$ <i>Offset</i> $= 0$	If instruction implies a read, the referenced pointer is pre-incremented. The Effective Address is the new stack pointer contents.  If the instruction implies a write, the Effective Address is the contents of the references stack pointer. The stack pointer is post-decremented.

Figure 2:  $\mu$ P Addressing ModesFigure 3:  $\mu$ P Instruction Format

### 3 Instruction Types

Once data flow strategy has been determined, the instruction set and addressing modes of the processor must be selected. Here, a wide variety of possibilities presents itself. Since the processor implemented is intended for an interrupt driven environment, it was decided that the machine should be stack oriented and provide good support for stack based operations. The addressing modes summarized in Figure 2 were finally decided upon. Direct referencing of memory locations with a pointer contained in the instruction itself provides simple access of memory mapped registers, global variables and targets for normal jump and jump subroutine instructions. The indexed with offset mode provides support for jump tables, arrays, stack oriented local variable access, and subroutine argument passage. The auto-decrement and increment modes support implied push and pop operations as a part of any instruction, ease the placing of arguments on the stack for passage to subroutines, and allow the return from subroutine instruction to be implemented as a special case of the jump instruction.

Figure 4 summarizes the instructions set which was selected. Each instruction can be operated in any addressing mode. The actual instruction format is shown in Figure 3.



Op Code	Mnemonic	Register	Description
0100	ld	a/b	Load Register
1111	st	a/b	Store Register
1000	jsr	—	Jump to Subroutine
0000	jmp	—	Absolute Jump
0010	and	a/b	Bitwise And
0011	or	a/b	Bitwise Or
1100	add	a/b	Addition
1110	sub	a/b	Subtraction
0110	not	a/b	Bitwise Complement
0101	xor	a/b	Bitwise Exclusive Or
1101	cmp	a/b	Skip next instr if not equal
1010	tst	a/b	Bitwise And, then skip next if 0
0111	shl	a/b	Shift Left
0001	shr	a/b	Shift Right
1001	lds	s/t	Load Stack Register
1011	ien	—	Enable/Disable Interrupts

Figure 4:  $\mu$ P Instruction Set

Since the arithmetic unit already provides an adder and a zero detect circuit for the implementation of the base instruction set, virtually no additional hardware in the datapath was required to implement the addressing modes. If a hardware multiplication instruction had been included in the instruction set, it would have been possible to utilize it during address generation to provide very sophisticated support for array accessing.

The requirement that all instructions be implemented with the same processing sequence places severe restrictions on the type of conditional statements that can be provided, however. A test and skip next instruction pattern was selected since it fits the required schema and was possible to implement without disturbing the pipelined flow of instructions. No retry of instructions is necessary, since the results of the test are always known in time to abort the effects of any subsequent instruction.

## 4 Implementation

### 4.1 General

The processor was designed using structured logic design techniques in a custom environment. High operating speeds and compact layouts were achieved through the extensive use of pass-logic.

The use of an orthogonally partitioned instruction set and an instruction invariant processing sequence resulted in extremely small and simple control circuitry. Consequently, the speed of a machine cycle is limited only by delays in the datapath— not by propagation

Cycle1:	$PC \xrightarrow{p} AR$	$AR \xrightarrow{p} AR$	
Cycle2:	$RAM \xrightarrow{m} MO$	$PC++$	
Cycle3:	$MO \xrightarrow{i;alu} Pipe2$	$SP(--) \xrightarrow{r;alu} Pipe1$	$0 \xrightarrow{r;alu} Pipe1$
Cycle4:	$Pipe1 \xrightarrow{add;o;p} AR$ $A/B/PC \xrightarrow{a} MI$	$Pipe2 \xrightarrow{add;o;p} AR$ $SP(--) \xrightarrow{p} AR$	$Pipe1/Pipe2 \xrightarrow{add;o} SP$
Cycle5:	$RAM \xrightarrow{m} MO$	$MI \xrightarrow{m} RAM$	
Cycle6:	$MO \xrightarrow{i;alu} Pipe2$	$A/B \xrightarrow{r;alu} Pipe1$	$0 \xrightarrow{r;alu} Pipe1$
Cycle7:	$Pipe1 \xrightarrow{add;o} A/B$	$Pipe2 \xrightarrow{add;o} A/B$	$Pipe1/Pipe2 \xrightarrow{add;o}$

Figure 5:  $\mu P$  Register Transfer Sequences

delays in the controller.

The processor itself is a simple design. Approximately three man months were required to design the circuit and verify its logical correctness through extensive logic simulations. During this time a software model of the processor was also written to aid logic verification and a macro-assembler was written for software development. Four man months were required to implement the layout and verify its correctness.

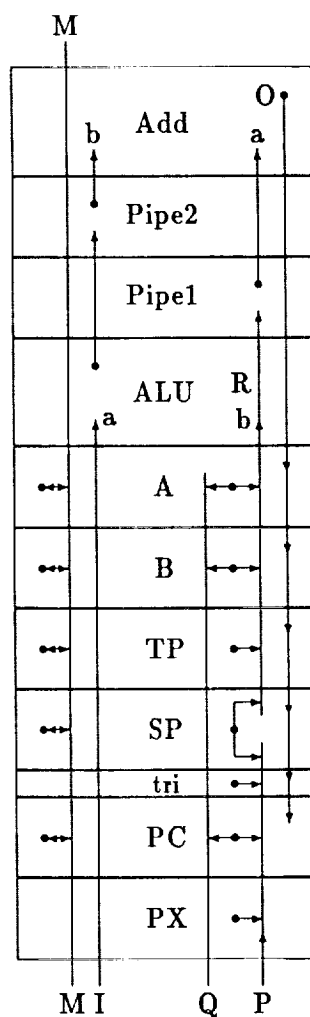
The processor was implemented in a  $1.6\mu m$  CMOS process and subsequently shrunk to a  $1.0\mu m$  process due to size considerations. It runs at a clock frequency of 28MHz under worst case processing assumptions, 140deg C junction temperature, and 4.1V internal supplies. The processor was completely functional on first silicon. Under typical conditions, it should run at nearly 60MHz, which corresponds to an instruction rate of 14MIPS worst case and 30MIPS typical. Currently, the limiting speed path is associated with the zero detect circuit. A redesign of this circuit would likely result in yet higher system performance.

## 4.2 Control

The upper bits the data bus are fed into the control section where they are pipelined parallel to the data passing through the datapath. The instruction decode and control of the datapath is simple, since both control and data are pipelined in an equivalent manner. The individual control lines to the datapath are decoded directly from the pipelined instructions. The logic of the control section fits on one C sized sheet of logic. It consists of four stages of pipeline registers, 61 NAND gates (most of which are 2 to 4 input gates), 10 NOR gates, and various inverter/buffers. It contains no state-machines except those required for interrupts and memory cycle stealing by the IO subsystem— and these are exclusively single bit state-machines.

## 4.3 Data Path

The datapath consists of a register stack and a pipelined Adder and ALU. Figure 6 is a signal flow diagram of the datapath. The M Bus is used for all memory mapped data

Figure 6:  $\mu$ P Register Stack

transfers. The P Bus drives the Address bus of the RAM through a clocked register, AR, located in the pads. The I bus is the data input bus from the memory, and the Q bus is the data bus to the RAM. The I and Q bus are combined into a single bi-directional bus at the chip pads through the MI and MO registers. (MI receives data from RAM during a read and MO outputs data to RAM during a write.) The datapath operates as follows. The instruction fetch address is driven from either the program counter or the secondary program register onto the P (address) bus. Two clock cycles later, the instruction arrives on the I bus, where it is fed through the ALU. At this time the Op Code portion of the instruction is stripped off and the remaining bits are used to form either an absolute address or an offset for the stack relative mode of operation. The results are clocked into the Pipe registers. Next clock cycle, this address/offset is either passed unaffected through the ADDER (absolute addressing mode) or added to the contents of the selected address register (SP or TP) (indirect addressing mode), and the results are driven onto the P (address) bus through a tri-state driver. If the instruction implies a write to memory,

the appropriate data is driven onto the Q bus from either A,B or PC. If the instruction implies a read the requested data enters the datapath via the I bus two clock cycles later and is processed by the ALU and ADDER in succession, at which time the results are loaded into the appropriate register. An RTL description of the data transfers comprising the data processing sequences utilized to implement the entire instruction set is shown in Figure 5. It should be noted again that though this processing sequence is seven clock cycles deep, processing of a new instruction starts every other clock cycle.

The ALU and adder are both implemented using pass logic. The ALU consists of a single cell replicated 16 time, each of which consists of only 23 n-channel pass gates and 9 inverter/buffers. The ALU performs all bitwise operations and provides a zero detect function which is used in the conditional skip instructions, as well as the detection of the auto increment/decrement addressing modes. The OpCode (figure 4) bit patterns were selected such that the upper bits of the instructions themselves become the control lines for the ALU with minimal remapping.

The configuration selected to implement the ADDER is a modified transmission gate conditional sum scheme [10]. The configuration is small, regular, and very fast.

## 4.4 IO Subsystem

Though not a primary topic here, it should be mentioned that a complete IO subsystem was implemented and integrated with the microprocessor described here. It consisted of a DMA subsystem which was responsible for the bulk transfer of data around the chip, two serial ports for low speed data transmission and acquisition, a parallel port for the transfer of data to and from an external microprocessor, as well as a prioritized interrupt/event passage system.

## 5 Conclusion

Present day integrated circuit fabrication processes support levels of integration adequate for the construction of on-board microprocessor based controllers which occupy only a small portion of the available circuit area. Such processors can be readily designed for different cost-performance tradeoffs, as required for specific applications. The outlay of engineering time need not be excessive and the use of high-level languages for code development makes the underlying instruction set transparent to the firmware developer, and eases code migration, development and support.

## References

- [1] A. Agrawal et.al., "The Scalable Processor Architecture (SPARC)," COMPCON '88 Proceedings, 1988, pp. 278-283.
- [2] H. Bakoglu, T. Whiteside, "RISC System/6000 Processor Architecture," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 8-15.

- [3] D. Curtin, L. Porter, *Microcomputers: Practices and Procedures*, Prentice-Hall, 1986.
- [4] G. Grohoski, J. Kahle, L. Thatcher, C. Moore, "Branch and Fixed-Point Instruction Execution Units," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 24-32.
- [5] P. Hester, "RISC System/6000 Hardware Background and Philosophies," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 2-7.
- [6] J. McLeod, "Tough Choices Ahead in Microprocessors," *Electronics*, May 1989, pp. 70-78.
- [7] *MC68020 32-Bit Microprocessor User's Manual*, 2nd Edition, ISBN 0-13-566860-3, Prentice-Hall, Englewood Cliffs, NJ, 1985, p. 1-6.
- [8] D. Patterson, C. Sequin, "A VLSI RISC," *IEEE Computer*, vol. 15, No. 9, Sep 1982, pp. 8-12.
- [9] C. Rowen et.al., "RISC VLSI Design for System Level Performance," *VLSI Systems Design*, March 1986, pp. 81-88.
- [10] A. Rothermel, et al., "Realization of Transmission-Gate Conditional-Sum (TGCS) Adders with Low Latency Time," *IEEE JSSC*, Vol. 24, June 1989, pp. 558-561.
- [11] R. Sherburne, M. Katevenis, D. Patterson, C. Sequin, "A 32b Microprocessor with a Large Register File," *Digest of IEEE International Solid-State Circuits Conference*, Feb 1984, pp. 168-169.

17. The Commission has also received information from the Government of the Republic of the Philippines that the Philippine National Police (PNP) has been conducting operations in the area of the conflict.

The Commission has also received information from the Government of the Republic of the Philippines that the Philippine National Police (PNP) has been conducting operations in the area of the conflict.

The Commission has also received information from the Government of the Republic of the Philippines that the Philippine National Police (PNP) has been conducting operations in the area of the conflict.

The Commission has also received information from the Government of the Republic of the Philippines that the Philippine National Police (PNP) has been conducting operations in the area of the conflict.

# A Modified Reconfigurable Data Path Processor

G. Ganesh, S. Whitaker and G. Maki <sup>1</sup>

NASA Space Engineering Research Center for VLSI System Design

University of Idaho, Moscow, Idaho 83843

Phone: 208-885-6500 Fax: 208-885-7579

**Abstract-** High throughput is an overriding factor dictating system performance. In this paper, a configurable data path processor is presented which can be modified to optimize performance for a wide class of problems. The new processor is specifically designed for arbitrary data path operations and can be dynamically reconfigured.

## 1 Introduction

High performance computers are increasingly in demand in areas of weather forecasting, structural analysis, etc.. These often require architectures which are different from the standard von-Neumann's machine also called the Standard Stored Program Computer. The stored program computers are designed to be general purpose and is not optimized for any specific problem. Fully customized architectures can be optimized to achieve maximum performance for a specific problem, but such processors cannot usually be adapted to produce solutions to different problems.

Modern technology opens new dimensions to the designer of high performance systems, by providing low cost VLSI modules which have high computational throughput. For a given functionality, there are two major dimensions of performance:- Delay and Throughput. High throughput is the most critical factor in real time processing of massive amounts of data, for example in Digital Signal Processing, Data Base operations, etc.. Since general purpose parallel computers cannot offer real time processing speeds, special purpose computers become the only appealing alternatives.

Special purpose processors can be of two types: 1) Dedicated Processors and 2) Reconfigurable/Programmable Processors. While the former are characterized by high processing speeds, inflexibility, long design time and high design cost, the latter have advantages of greater flexibility in coping with changes in the object problem, system specification and greater design economy with some reduction in throughput.

This paper presents a general purpose accelerator which is an enhancement over [1], that allows a variety of data path configurations, each characterized by its own topology of activated interconnections and hence applicable to a wide range of applications.

This configurable architecture combines the general purpose advantages of the stored program machine with the optimization of a fully customized architecture to achieve maximum performance for a broad class of problems. Every functional unit, data path and

---

<sup>1</sup>This research was supported ( or partially supported ) by NASA under Space Engineering Research Center Grant NAGW-1406.

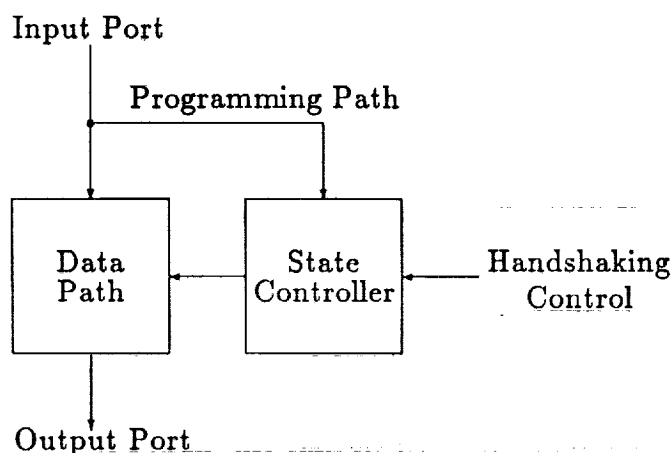


Figure 1: Block Diagram

control structure can be individually optimized for a given algorithm. The architecture presented is capable of operating in parallel, pipelined or sequential modes. The user configures the data path through programming. The architecture can be altered during operation by reprogramming or can be initialized and fixed for dedicated processing or can be attached to a host processor.

The reconfigurable processor differs from the stored program computer in the sense that there is no instruction fetch-decode-execute cycle. Moreover, an operation can be executed every clock pulse in every data path element.

## 2 Processor Design

The data path and the control structure have been designed to allow sequential, pipelined or parallel operation. The processor is configured as a set of identical data path elements with an overall controller. The top level block diagram of this processor is shown in Figure 1. There are two major components: the data path, which is an ALU-register stack to manipulate the data, and the state controller, which controls the register stack. The actual hardware configuration of the data path is specified during the programming of the State controller.

### 2.1 Data Path

Each data path element is as shown in Figure 2. Let there be  $m$  data path elements, each  $n$  bits wide. Direct communication between each data path element is an essential feature to achieve pipeline or parallel operation. Therefore, to allow all possible register to register communications, the data path bus must be  $m \times n$  bits wide. This complete connectivity results in the flexible reconfigurability, but also limits the number of data path elements.

Each data path element consists of a Multiplier Accumulator (MAC) which multiplies two eight bit numbers and also adds two sixteen bit numbers to the product.  $(a.b+c+d)$ . This output is stored in a globally accessible register of the data path element. Also



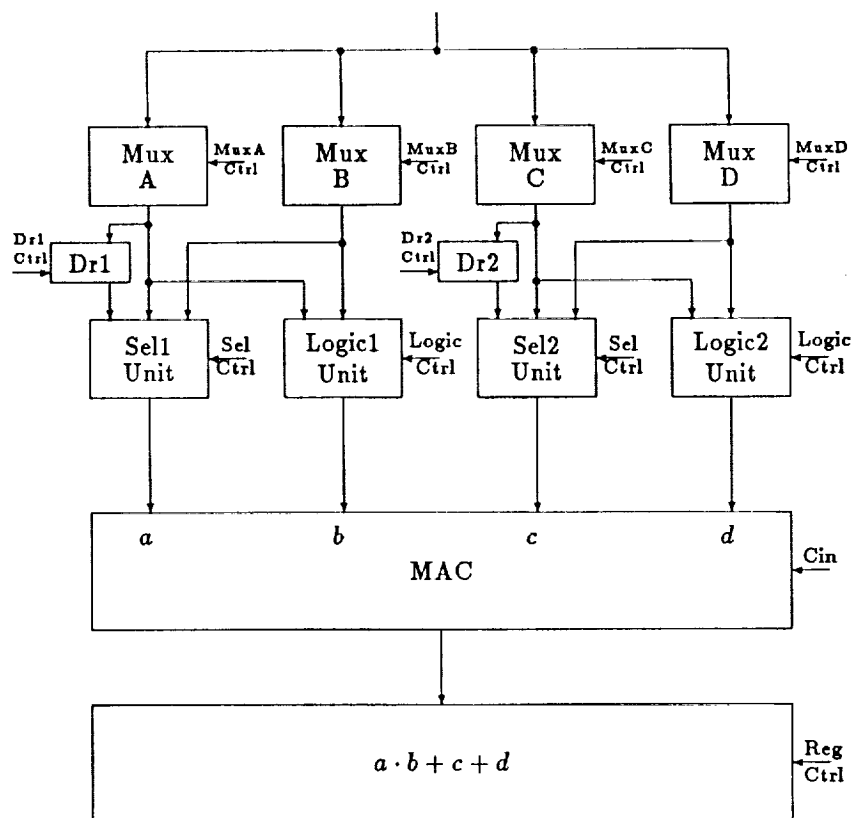


Figure 2: Data Path Element

contained within each data path element are two dedicated registers, which are used for operations local to that data path element. The addition of these dedicated registers is one of the improvements over [1]. This avoids the use of an entire data path element for the purpose of storage only. Since the area of a data path element is constrained by the  $m \times n$  interconnect bus the addition of these registers should have little impact on the overall chip area.

The data path also contains a set of ALU and selector units. The ALU can implement an arbitrary arithmetic/logic operation. The operations of the first logic unit is as shown in Table 1. The selector unit selects the output of its respective multiplexors or the output of the respective dedicated register as shown in Table 2. The  $m$  to 1 multiplexors can select the output of any of the  $m$  globally accessible registers. The MAC operates on the output of the selector unit and the logic unit to allow a mixture of arithmetic and logic functions. Table 3 shows example of ALU operations that can be performed. CI is the carry in data bit.

Each globally accessible register is controlled as defined in Table 4. The dedicated registers are controlled as shown in Table 5.

The control word for each data path element structure is shown in Figure 3. For 16 data path elements, the control word is 33 bits wide.

Logic Control	Logic Operation
0 0 0 0	0
0 0 0 1	A AND B
0 0 1 0	A AND B'
0 0 1 1	A
0 1 0 0	A' AND B
0 1 0 1	B
0 1 1 0	A XOR B
0 1 1 1	A OR B
1 0 0 0	A NOR B
1 0 0 1	A XNOR B
1 0 1 0	B'
1 0 1 1	A' NAND B
1 1 0 0	A'
1 1 0 1	A NAND B'
1 1 1 0	A NAND B
1 1 1 1	1

Table 1: Logic Unit 1 Control

MC	Selector Output
00	A Mux
01	B Mux
10	Dr1
11	0

Table 2: Selector Unit 1 Control

Logic Unit	Sel Unit	CI	Output
0 0 0 0	A	1	A + 1
1 1 1 1	A	0	A + 1
0 0 1 1	-	1	A + 1
0 1 0 1	A	0	A plus B
1 0 1 0	A	0	1's complement A - B
1 1 0 0	-	1	2's complement A
1 0 1 0	A	1	2's complement A - B

Table 3: Example ALU operations

MuxA Ctrl	MuxB Ctrl	MuxC Ctrl	MuxD Ctrl	Logic1 Ctrl	Logic2 Ctrl	Sel1 Ctrl	Sel2 Ctrl	Reg Ctrl	C I	Dr1 Ctrl	Dr2 Ctrl									
32	29	28	25	24	21	20	17	16	13	12	9	8	7	6	5	4	3	2	1	0

Figure 3: Data Path Element Control Word

RC1	RC2	Register Function
0	0	Hold Present Data
0	1	Load MAC Output
1	0	Shift MAC Right and Load
1	1	Shift MAC left and Load

Table 4: Global Register Load Control

Dr1	Register Function
0	Hold Present Data
1	Load Mux Output

Table 5: Dedicated Register Load Control

## 2.2 Control

The state controller specifies the control words for each data path element. The hardware compiled control words are contained in a control store memory as depicted in Figure 4. The output of each word from the control store drives each data path element. A total of 536 bits are needed in each control store word to control the data path elements in a 16 element, 16-bit data path structure. Program control within the control store is implemented with a program location counter. The control store can be of an arbitrary depth; here, it is depicted as 256 words deep. To perform a jump within the control store, an 8-bit jump address is provided in each control store word as depicted in Table 6.

The control store must be specified prior to operation. This specification (hardware compilation) can be achieved through the input port, 16 bits at a time. After the control store is specified, the processor is ready to operate in real time.

## 3 Operation

The control store word defines the operation and the source of data (registers) for each data path element. The output of any pair of registers  $R_i$  and  $R_j$ ,  $i, j = 0, 1, 2, \dots, 15$  can be input to a data path element. In general, the operation can be specified as

$$R_i[ALUoperation]R_j \rightarrow R_k \quad (1)$$

33 bits	33 bits	...	33 bits	8 bits
Data Path Control Word Cell 0	Data Path Control Word Cell 1	Data Path Control Word Cell i	Data Path Control Word Cell 15	Program Counter Address

Table 6: Control Store Word

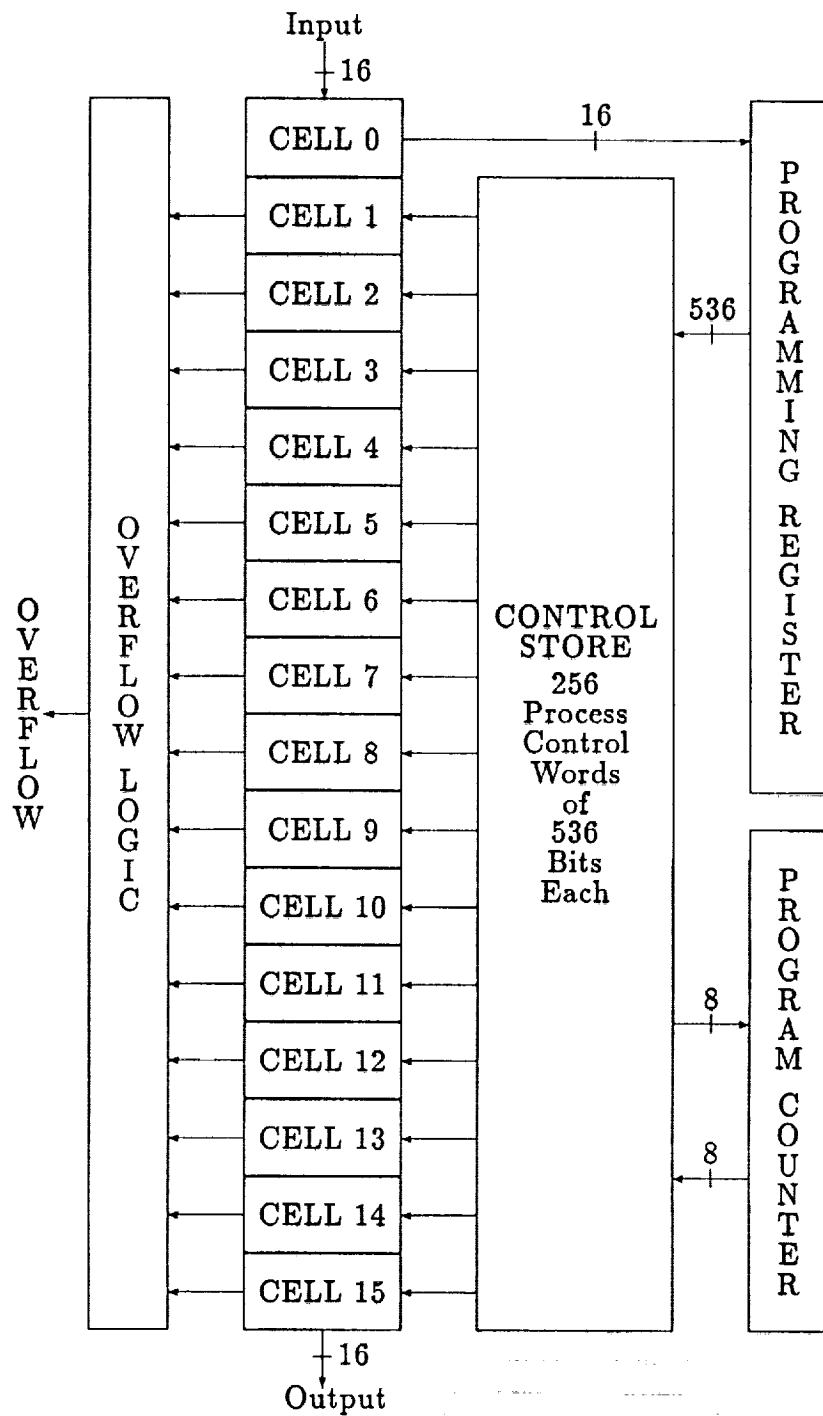


Figure 4: Control Store and Data Path

which means that the result of an ALU operation upon the contents of any register pair  $R_i$  and  $R_j$  can be placed into register  $R_k$ . This is true for any and all registers in the data path and all operations occur simultaneously. Since each data path element can function as an independent element, the entire data path can be configured to operate in the sequential, pipelined or parallel modes. The controller also specifies the next state of the controller and provides handshaking for external input and output control functions. The memory can be ROM for dedicated processing or RAM or EPROM where field programmability is desired. Depicted in Figure 4 is a feature where the control store can be programmed via the input data port. The entire control store can be initialized in a 16 bit word serial manner.

The control store is specified prior to operation. Once the control store is specified, the processor executes at the rate specified by the system clock. With static cells, the system clock can range from d.c. to the maximum allowable by the IC process.

### 3.1 Examples

Consider the following Digital Filter examples to illustrate the use of this processor. The general second order difference equation is

$$y(n) = \alpha_0 x(n) + \alpha_1 x(n-1) + \alpha_2 x(n-2) - \beta_1 y(n-1) - \beta_2 y(n-2). \quad (2)$$

This implements an IIR filter. For an FIR filter the equation simplifies to

$$y(n) = \alpha_0 x(n) + \alpha_1 x(n-1) + \alpha_2 x(n-2). \quad (3)$$

To implement the FIR filter in the architecture presented in this paper, let  $Dr_{61}$  contain  $\alpha_0$ ,  $Dr_{51}$  contain  $\alpha_1$  and  $Dr_{41}$  contain  $\alpha_2$  as shown in the simplified block diagram of Figure 5. Also let  $R_4$ ,  $R_5$ ,  $R_6$  and  $R_0$  be initially reset. The operations can be described in a register transfer language where each  $P_i$  is a control state that defines the data transfers that take place when  $P_i$  is active.

$P_0$ :  $Data \rightarrow R_0$

$P_1$ :  $R_0 \cdot Dr_{61} + R_5 \rightarrow R_6$ ,  $R_0 \cdot Dr_{51} + R_4 \rightarrow R_5$ ,  
 $R_0 \cdot Dr_{41} \rightarrow R_4$

Assuming that constants are preloaded into the registers and that 2's complement arithmetic is used, the control word for each data path element ( $R_i$ ) is shown in Table 7. Each control state  $P_i$  represents one parallel control word; the portion of the control word for each  $R_i$  is shown on a series of lines for the sake of simplicity. Register control for all other registers not shown in Table 7, the register control bits in their control words are 00, indicating no operation, for that control state,  $P_i$ .

There are a total of 5 operations that occur in 2 clock pulses. If this processor operated at 20 MHz, 50 million operations per second would be performed.

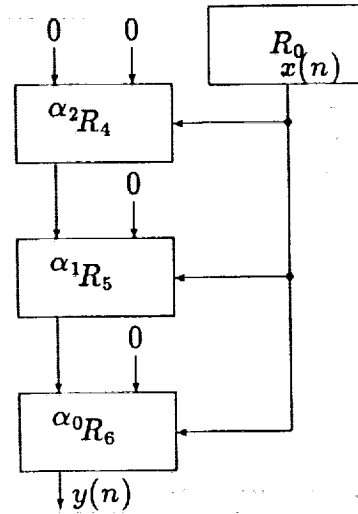


Figure 5: FIR Filter Block Diagram

State	Reg	MuxA	MuxB	MuxC	MuxD		
$P_0$	$R_0$	A	-	-	-		
$P_1$	$R_4$	$R_0$	-	-	-		
	$R_5$	$R_0$	-	$R_4$	-		
	$R_6$	$R_0$	-	$R_5$	-		
State	Reg	ALU1	ALU2	SC1	SC2	CI	RC
$P_0$	$R_0$	1111	0000	00	11	0	01
$P_1$	$R_4$	0011	0000	10	11	0	01
	$R_5$	0011	0000	10	00	0	01
	$R_6$	0011	0000	10	00	0	01

Table 7: FIR Filter Control Word Programming

State	Reg	MuxA	MuxB	MuxC	MuxD		
$P_0$	$R_0$	A	-	-	-		
$P_1$	$R_2$	$R_0$	-	$R_6$	-		
	$R_6$	$R_0$	-	$R_7$	-		
	$R_7$	$R_0$	-	-	-		
	$R_{10}$	$R_9$	-	-	-		
	$R_9$	$R_9$	-	$R_2$	$R_8$		
	$R_8$	$R_9$	-	-	-		
State	Reg	ALU1	ALU2	SC1	SC2	CI	RC
$P_0$	$R_0$	1111	0000	00	11	0	01
$P_1$	$R_2$	0011	0000	10	00	0	01
	$R_6$	0011	0000	10	00	0	01
	$R_7$	0011	0000	10	11	0	01
	$R_{10}$	1111	0000	00	11	0	01
	$R_9$	0011	0101	10	00	0	01
	$R_8$	0011	0000	10	11	0	01

Table 8: IIR Filter Control Word Programming

$Dr_{21}$	$\alpha_0$
$Dr_{61}$	$\alpha_1$
$Dr_{71}$	$\alpha_2$
$Dr_{91}$	$\beta_1$
$Dr_{81}$	$\beta_2$
$R_{10}$	$y(n)$
$R_9$	$y(n-1)$
$R_8$	$y(n-2)$

For an IIR filter, consider the following register assignment. A register transfer language description of the operations to implement the IIR filter equation would be

$P_0$ :  $Data \rightarrow R_0$

$P_1$ :  $R_0 \cdot Dr_{21} + R_6 \rightarrow R_2, R_0 \cdot Dr_{61} + R_7 \rightarrow R_6,$

$R_0 \cdot Dr_{71} \rightarrow R_7, Dr_{81} \cdot R_9 \rightarrow R_8,$

$R_2 + R_8 + R_{91} \cdot R_9 \rightarrow R_9, R_9 \rightarrow R_{10}.$

Assuming again that constants are preloaded into the registers as 2's complement numbers and that 2's complement arithmetic is used. The control word for each data path element is shown in Table 8. There are a total of 9 operations that occur in 2 clock pulses; operating at 20 MHz, 90 million operations per second would be performed.

## 4 Summary

A new architecture has been presented which allows for sequential, pipelined, or parallel operation. A control-data path structure consists of  $m$  identical data path elements. The data path elements can be independently specified to allow parallel or pipelined operation. The control of the data path is specified by the control store memory. The processor can be

a dedicated stand alone machine or attached to a general purpose processor. As an attached processor, it can be dynamically modified to assume different data path configurations if the control store is RAM based. It is proposed that this architecture is a first step in producing a processor that allows the digital designer the same kind of flexibility in altering data path configurations as field programmable gate arrays offer alternatives to the logic designer.

**Acknowledgement** This research was supported in part by NASA under grant NAGW-1406 and grant NAG5-1043.

## References

- [1] G. Maki, S. Whitaker and G. Ganesh, "A Reconfigurable Data Path Processor", Proceedings of the IEEE ASIC Conference, Rochester NY, Sept., 1991.



# Systolic Array IC for Genetic Computation

D. Anderson  
Hewlett Packard  
Northwest IC Division  
1020 NE Circle Blvd.  
Corvallis, OR 97330  
daryl@hpcvcan.cv.hp.com

## 1 Introduction

Measuring similarities between large sequences of genetic information is a formidable task requiring enormous amounts of computer time. Geneticists claim that nearly two months of CRAY-2 time are required to run a single comparison of the known database against the new bases that will be found this year, and more than a CRAY-2 year for next year's genetic discoveries, and so on.

The DNA IC, designed at HP-ICBD in cooperation with the California Institute of Technology and the Jet Propulsion Laboratory, is being implemented in order to move the task of genetic comparison onto workstations and personal computers, while vastly improving performance.

The chip is a systolic (pumped) array comprised of 16 *processors*, control logic, and global RAM, totaling 400,000 FETS. At 12 MHz, each chip performs 2.7 billion 16 bit operations per second. Using 35 of these chips in series on one PC board (performing nearly 100 billion operations per second), a sequence of 560 bases can be compared against the eventual total genome of 3 billion bases, in minutes — on a personal computer.

While the designed purpose of the DNA chip is for genetic research, other disciplines requiring similarity measurements between strings of 7 bit encoded data could make use of this chip as well. Cryptography and speech recognition are two examples.

A mix of full custom design and standard cells, in CMOS34, were used to achieve these goals. Innovative test methods were developed to enhance controllability and observability in the array. This paper describes these techniques as well as the chip's functionality.

This chip was designed in the 1989-90 timeframe.

## 2 Goals

The main project goal was to produce a device, for a larger system, that would prove the new computing architecture. This meant integrating as much functionality as was reasonable, with respect to cost. This includes as many processors, as much RAM per processor, and as many other desired functions as possible. Performance was a lesser concern, largely due to disk access being the initial system performance limiter, and also because the architecture provides the main performance breakthrough. Limiting power dissipation was a lesser, but real concern as well.

At the outset of the project, a standard cell implementation was envisioned that might contain 10 processors, each with 32 bytes of RAM on a 1 cm square device. In the end, a custom solution provided 16 processors, each with double the functionality and 128 bytes of RAM, on a roughly 1 cm x 1.2 cm device.

### 3 Functionality

The primary function of the system, comprised largely from a series of DNA chips, is to locate regions of similarity between strings of genetic bases, represented in ASCII by the characters; A, C, G, and T. A terse description of the method for achieving this end, is as follows.

First, the *primary string* is converted by an external processor, such that each character in that string is replaced by four *match scores*, one for each of the four possible characters that it may be compared against in the *secondary string*. These scores, for each character in the primary string, are loaded into the local RAMs of each successive processor, such that each RAM contains the four bytes representing the four possible scores caused by interaction of characters in the secondary string with that single character of the primary string. Each processor's RAM is 128 bytes, enough to accommodate full ASCII. Now, each processor behaves as the agent of one character in the primary string; hence, the length of the primary string is initially limited to the number of processors in the system (16 x number of DNA chips). Through software the length can be expanded without limit, by method of partitioning the string and using sufficient overlap.

Secondly, a number of constants are loaded into each chip by the external system processor, such as; chip location within the pipeline, how to deal with gaps that naturally occur within genetic sequences, and others.

At this point, the pipeline begins to function. The secondary string enters the front of the pipeline and is passed from one processor to the next on each successive clock. Each character within this string is used as an address to the local RAM of the current processor visited by that individual character. By this method, the appropriate score is retrieved from the local RAM for the interaction between the characters of the two separate strings. Along with the former occurring, the following three equations are processed within that same clock cycle, in each processor. (Smith and Waterman, *Best Subsequence Alignments Algorithm*.):

$$H_{i,j} = \max\{0, H_{i-1,j-1} + s(a_i, b_j), E_{i,j}, F_{i,j}\}$$

with

$$E_{i,j} = \max\{H_{i,j-1} - (u_E + v_E), E_{i,j-1} - v_E\}$$

$$F_{i,j} = \max\{H_{i-1,j} - (u_F + v_F), F_{i-1,j} - v_F\}$$

where  $F$ ,  $H$  and  $b$  are pipelined,  $E$  is fed back within the processor,  $u_E, v_E, u_F$ , and  $v_F$  are constants dealing with sequence gaps, and  $s(a_i, b_j)$  is the score produced by the intersection of the two characters from the different strings.

Additionally, each processor monitors its  $H$  value, which represents quantitatively the similarity between strings, and detects its peak. If this peak exceeds a programmed threshold, this value, as well as the location of this occurrence within the secondary string, is piped along through the remaining processors on the given chip and then stored in the chip's global FIFO RAM. The range of this location value limits the secondary string to 4 million characters. However with use of external software, an unlimited string can be applied.

This process occurs simultaneously in all processors on each chip, until the entire secondary string has been piped all the way through, or the external system processor interrupts. The equations and peak detector are implemented with five adders and seven comparators; the values  $(u_E + v_E)$  and  $(u_F + v_F)$  are provided as constants.

When a value has been stored in the global FIFO RAM, the chip signals the external system processor, and at the system processor's convenience, reads that data from the chip into a global system RAM. This is the raw similarity information desired from the system. Of course, if any chip's FIFO nears overflow, a system interrupt is issued, by that chip, to pause the entire pipeline until the RAMs can be emptied.

## 4 Design Challenges

Technical challenges included; performance, power, and density concerns, as well as problems pertaining to pad switching noise and testability.

By custom designing most circuitry for near maximum density, lower power and higher performance fell out as by-products. Most N channel devices in the pipeline were sized at  $5\mu$  wide and  $1\mu$  long. The small devices reduced power consumption, as well as greatly improved the circuit density. By careful floorplanning to minimize interconnect capacitance, chip performance was improved over that of a standard cell approach. One of the key sub-modules within the processor is a 16 bit adder. At  $426\mu$  by  $215\mu$  (896 FETs), the custom adder is one seventh the size of its standard cell implementation; at 4 mW, its power consumption is one sixth; and at 11 nS, its performance is improved by more than two fold over the standard cell solution.

While the conservative design goal of 12 MHz does not seem worthy of CMOS34, consider two of the paths to be traversed in the 83 nS cycle; 1) Register — 16 bit signed addition — 6 x (16 bit signed compare and select greatest) — 5 gates — Register, and 2) Register — address RAM — 16 signed bit addition — 3 x (16 bit signed compare and select greatest) — 5 gates — Register.

The next area of concern was with pad switching noise. This resulted from being bound to a 208 lead Quad Flat Pack, with 190 signal pins, leaving only 18 power pads. While having a full synchronous design helped in some aspects, it also created the possibility of having all 65 pipeline outputs and all 32 global data bus pads switch simultaneously. Additionally, several other system pads could be switching as well. It is helpful that all pipeline input signals are latched on the rising clk, while the pipeline outputs do not change until a number of gate delays later. However, several volts of supply noise could easily be

generated by switching the standard pads, causing erroneous inputs and outputs on the global system pads. Additionally, latchup was a concern.

The solution was to create three modified pads and use an expanded power distribution scheme. All pad input sensors (TTL level Schmitt) were connected to one Vdd pad and two Gnd pads; 124 inputs total. The output drivers for pipeline outputs, and global data bus were connected to 4 Vdd pads and 5 Gnd pads; 97 outputs total. The remaining 4 global system pads, capable of causing system interrupts, were connected to their own isolated pair of power pads. Lastly, the chip core and output pad stage-ups were placed on two pairs of power pads.

While this helped to isolate the noisy circuitry from sensitive circuitry, the noise spikes on the *dirty* power bus from output switching, were still too high. Several things were done to help reduce the noise. First, the drivers for the 65 pipeline outputs were greatly reduced so that the rise time on the Sentry 15's 60 pF load would be 40 nS worst case. These outputs will normally see only 7-10 pF in the product, as the output pad communicates only with the neighboring chip's input pad.

The global data bus pads created another problem in that their loading depended directly on how many DNA chips were placed in the system, as they all connect directly to one another. In the initial system, this load would be 275 pF. Since the 32 data bus pads were by far the largest contributor to noise, and because their load could vary, another scheme was employed. The data pads each contain two sets of output drivers; one small and one large. A signal to the pad determines whether the large drivers are used in parallel with the small ones, or whether the small drives are used alone. A control register bit is used to turn off the larger drivers, in the event that the data bus had a small capacitive loading, or that the noise from the larger drivers was simply unacceptable (in which case, the system clk rate would have to be reduced). The rise time for a 275 pF load with the large drivers is 20 nS, worst case, and 75 nS without those drivers.

Additionally, care was taken to turn on output drivers slowly; about a 3 nS to 4 nS rise time on the driver's gate. Skewing of data to the pad drivers also helped to reduce the switching noise.

The last major challenge was in the area of test. Standard methods for testing the part in its normal operating mode were seen to be near impossible. The controllability and observability of nodes deep in the pipeline of 16 processors was very near zero. Since each processor interfaces to the previous processor through a register bank, scan testing seemed to be the obvious solution. However, with about 150 register bits in each processor, and a total of 16 processors and one additional pipeline register buffer, the full scan vector length would be over 2500 bits. With a Sentry 15 limited to 256k total vectors, this would provide only 100 scan vectors, with no vector memory available for testing the 22k bytes of RAM, nor the control logic. Several thousand scan vectors were desired for testing the processors.

The solution was to take advantage of the fact that all of the processors are identical, and therefore given the same input scan vector, will produce the exact same resultant output vector in the register bank of the pipeline's next stage. The method then, is to scan in a vector that is only one processor register bank long (150 bits), into all 16

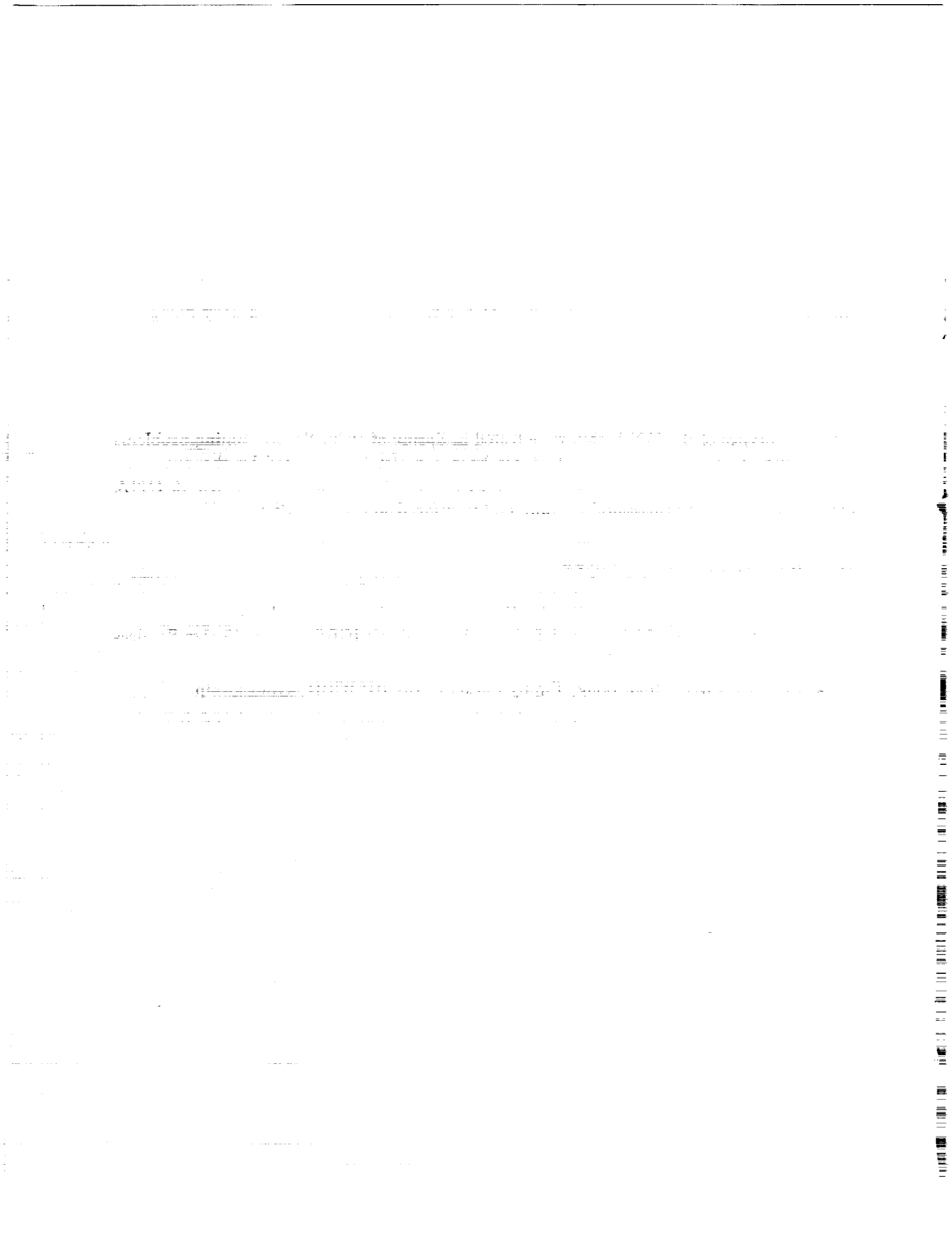
processors simultaneously. After clocking the device once in normal mode, as in standard scan methodology, the 16 resultant vectors are scanned out of the processors and onto 16 independent lines of the global data bus, readable from the chip's data bus pads. Additionally for testing in the product, all 16 scan outputs are connected, on chip, to a equality function. If, when in test mode, all of the 16 scan outputs are not equal, then an error pin is activated for notification of the external system processor. The system processor can then set another pin on the errant chip so that the pipeline data coming on chip is diverted around the 16 processors, to the final buffer register, thereby fixing the whole pipeline at the cost of those 16 processors.

## 5 Results

First prototypes of the **DNA** chip were tested in Spring of 1990. Several timing problems were found in chip functions that had not been completely simulated by the designers. Second prototypes produced perfect parts. JPL currently has a circuit board 16 **DNA** chips (a total of 256 processors) running and interfaced to a workstation.

## 6 Acknowledgments

- Ed Chin and Mike Yoo of JPL for their design work on the **DNA** chip.  
Tim Brown, Paul Liebert, and Bobbie Manne of HP-ICBD Design Center Services for helping with artwork generation.
- Joe Casprowiak, Joan Long, Jimmy Packer of the HP-ICBD layout group.
- Fred Perner and Lynn Roylance of HP Labs for initial work on the **DNA** investigation.



## High-Performance Multiprocessor Architecture for a 3-D Lattice Gas Model <sup>1</sup>

F. Lee, M. Flynn and M. Morf  
Computer Systems Laboratory  
Stanford University, Stanford, CA 94305

**Abstract-** The lattice gas method has recently emerged as a promising discrete particle simulation method in areas such as fluid dynamics. We present a very high-performance scalable multiprocessor architecture, called ALGE, proposed for the simulation of a realistic 3-D lattice gas model, Hénon's 24-bit FCHC isometric model. Each of these VLSI processors is as powerful as a CRAY-2 for this application. ALGE is scalable in the sense that it achieves linear speedup for both fixed and increasing problem sizes with more processors.

The core computation of a lattice gas model consists of many repetitions of two alternating phases: *particle collision* and *propagation*. *Functional decomposition by symmetry group* and *virtual move* are the respective keys to efficient implementation of collision and propagation.

### 1 Introduction

High performance computing has become a vital enabling force in the conduct of science and engineering research and development. In particular, simulations based on computational fluid dynamics are less costly and much faster than complex wind tunnel tests. In the past few years, the lattice gas method [3] has emerged as an attractive, robust and promising discrete particle simulation method for fluid flow simulations with complicated boundary conditions, that are difficult or impossible to solve with other methods. Various standard fluid dynamical equations, including the Navier-Stokes equations, can be obtained from lattice gas models after proper limits are taken [4].

The core computation of a lattice gas model is inherently suitable for execution on scalable parallel computing systems, without requiring floating point operations. Increasing amounts of computing power is needed to solve large scale simulation problems. It is believed that simple and practical application-specific computers (or co-processors) can achieve performance orders of magnitude higher than existing "general-purpose" supercomputers, that invariably focus on floating point operations. This belief has been confirmed in the case of two-dimensional simulation, but not in the case of three-dimensional simulation, which is much more important and challenging.

All existing special-purpose lattice gas computers such as CAM-6 [12], RAP1, RAP2 [1], and LGM-1 [6], deal with two-dimensional lattice gas models. Until today, only one other design, CAM-8 [10], proposed by Margolus and Toffoli, attempts to deal with three-dimensional models, but this proposal is limited to 16 or fewer state bits per lattice node.

---

<sup>1</sup>This work was supported by NASA Ames Research Center under contract NAGW 419.

Yet we need to simulate models with 24 bits or more per node in order to achieve realistic results in studying complex phenomena such as turbulent flow [2]. ALGE is to our knowledge the first special-purpose machine proposed to tackle a realistic 3-D lattice gas model.

## 2 Lattice Gas Models

In order to keep this paper self contained, we repeat some of the material from our previous publication [7], on which this paper is based.

In a lattice gas model, space and time are discretized. Time is divided into a sequence of equal time steps, at which particles reside only at the nodes of the lattice. The evolution consists of two alternating phases: (i) *propagation*: during one time step, each particle moves from one node to another along a link of the lattice according to its velocity; (ii) *collision*: at the end of a time step, particles arriving at a given node collide and instantaneously acquire new velocities. The properties of the lattice not only govern the propagation phase, but also significantly constrain the collision phase, because the *collision rules* must have the same symmetries as the lattice [4].

The *state* of a node can be denoted by the bit vector  $\mathbf{b} = (b_1, \dots, b_n)$ , where  $b_i = 1$  if a particle with the corresponding velocity  $\mathbf{v}^i$  is present<sup>2</sup>, and  $b_i = 0$  otherwise. Let  $\mathbf{b}(\mathbf{x}, t)$ , and  $\mathbf{b}'(\mathbf{x}, t)$  be the states of the node at position  $\mathbf{x}$  and time  $t$  before and after the collision respectively. The collision phase specifies that, for all  $\mathbf{x}$  and  $t$ ,

$$\mathbf{b}'(\mathbf{x}, t) = \mathcal{C}(\mathbf{b}(\mathbf{x}, t)) \quad (1)$$

where  $\mathcal{C}$  is a deterministic or non-deterministic  $n$ -input  $n$ -output boolean collision function. The propagation phase specifies that, for all  $\mathbf{x}$  and  $t$ ,

$$b_i(\mathbf{x} + \mathbf{v}^i, t + 1) = b'_i(\mathbf{x}, t) \quad (2)$$

An obstacle such as a plate, a wedge or an airplane wing is decomposed into a series of continuous links which approximate its geometrical shape. At nodes which represent an obstacle, particles are either bounced back or undergo specular reflection. This can be handled by adding one or more obstacle bits to the state of a node and adjusting the collision function appropriately.

Before simulation, the states of the nodes are initialized according to the initial distribution of particle densities and velocities. After simulation, nodes within a volume of tens of nodes on each side are averaged to compute the macroscopic density and momentum.

There are two types of boundary conditions on the lattice edges we are concerned with. The first type is the *periodic boundary condition*: the particles exiting from one edge are reinjected into the other edge in the same direction. The second type, related to a wind-tunnel experiment, consists in providing a flux of fresh particles on one side of the lattice and allowing an output flux on the other side. In this paper, we focus on the first type of

<sup>2</sup>In this paper, Roman and Greek indices refer respectively to labels and components.



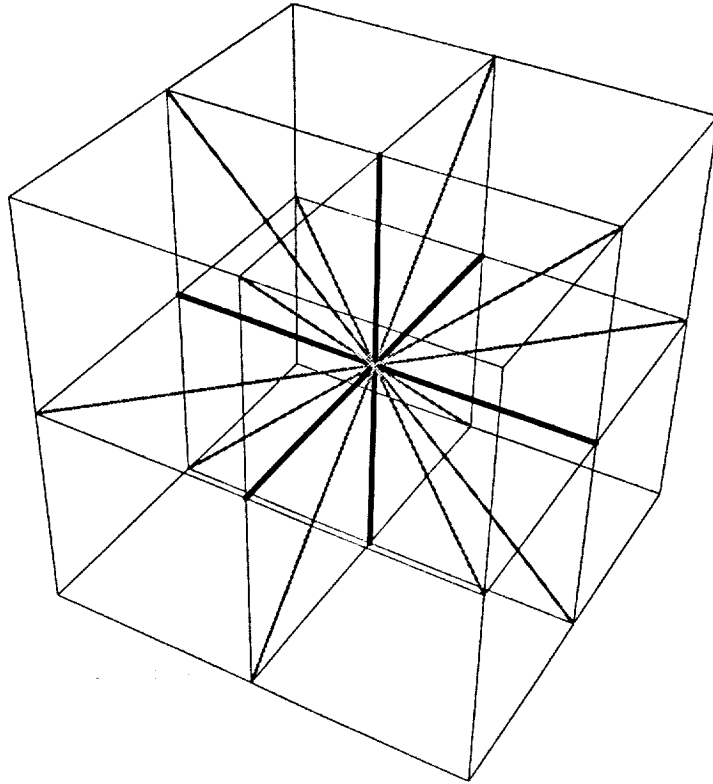


Figure 1: The pseudo-four-dimensional FCHC model. Only the neighbors of one node are shown as connected.

boundary condition, because it is basic: it requires no special treatment for nodes on the *edges*, as there are no edges in a wraparound lattice space. The second type can be dealt with as a simple extension.

## 2.1 Three-Dimensional Lattice

The particular lattice we are most interested in is the FCHC lattice used in three dimensional simulations [4,11]. A FCHC (face-centered hypercubic) lattice consists of those nodes, which are the points with signed integer coordinates  $(x_1, x_2, x_3, x_4) = \mathbf{x}$  such that the sum  $x_1 + x_2 + x_3 + x_4$  is even. Each node  $\mathbf{x}$  is linked to its 24 nearest neighbors  $\mathbf{x}'$  such that the vector  $\mathbf{x}' - \mathbf{x}$  corresponds to one of the following 24 values:

$$\begin{aligned} &(\pm 1, \pm 1, 0, 0), (\pm 1, 0, \pm 1, 0), (\pm 1, 0, 0, \pm 1), \\ &(0, \pm 1, \pm 1, 0), (0, \pm 1, 0, \pm 1), (0, 0, \pm 1, \pm 1). \end{aligned} \quad (3)$$

These 24 nearest neighbors form a regular polytope. With time steps normalized to 1, the vectors in (3) are also the 24 possible velocities of the particles arriving at or leaving from a node.

The pseudo four-dimensional FCHC model is derived by projecting the four-dimensional FCHC lattice to three dimension so that the fourth dimension has a periodicity of 1. Each node of a regular cubic lattice is a node in the model. Figure 1 shows the neighborhood of a node: along the gray links, connecting to 12 neighbors, at most one particle can propagate, with component  $v_4 = 0$ ; along the thick black links, connecting to 6 neighbors, up to two particles can propagate, with components  $v_4 = \pm 1$ .

## 2.2 Isometric Collision Rules

Associated with the FCHC lattice is the *isometry group*  $G$  of order 1152. Roughly speaking, an isometry is a symmetry operation such as rotation and reflection about the origin.

The isometric collision rules [5] require that

1. Every collision is an isometry: the output velocities are images of the input velocities in an isometry.
2. The isometry depends on the momentum only: the momentum of the input state is computed, and then normalized by taking advantage of the symmetries, and finally used for classification.
3. The isometry is randomly chosen among all optimal isometries: this is why non-determinism comes into play. (An optimal isometry is one which minimizes the viscosity of the lattice gas, so that higher Reynolds numbers can be reached.)

## 3 System Overview

This is an updated version of the design of ALGE as presented in [7]. The machine is organized as an array processor, which serves as a special purpose high performance computing engine to a host computer. The host computer downloads the problem (data) into the engine and offloads the engine-produced solution. The host provides the user interface to the computing engine and performs the pre-processing and post-processing phases of the simulation.

Figure 2 shows a 4x4 configuration of ALGE. The processors are connected as the nodes of a 2-D toroid. Each identical processor <sup>3</sup> (P) has its own local memory (M). In a simulation the 3-D problem space is decomposed into non-overlapped equal-sized partitions such that nodes with the same Z-coordinates map to the same memory space, and adjacent partitions map to adjacent memory spaces.

## 4 Processor Architecture

Figure 3 shows the functional block diagram of the processor. The processor contains the following units: several collision units, an address generator, a transposer, a switch, a

<sup>3</sup>It may contain several processing elements (PE) as referred in [7].

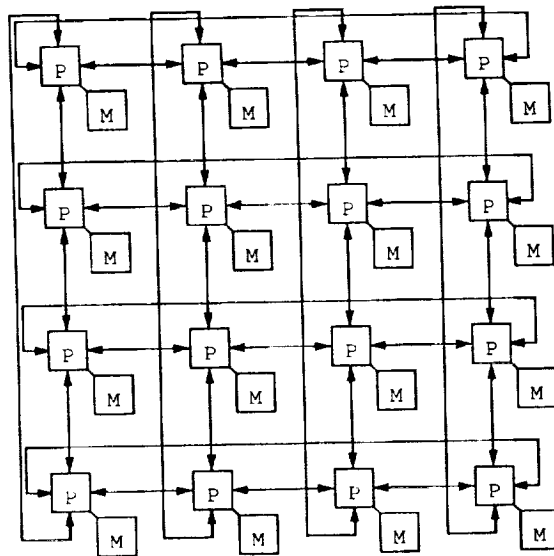


Figure 2: A 4x4 configuration of ALGE

memory address register (MAR), a memory data register (MDR) and a control.

The collision units can be viewed as the “arithmetic” units of a “superscalar” processor. Each unit is capable of computing one collision function per cycle. The address generator contains a register file and some modified adders. It is responsible for generating the proper address sequences for reading and writing data from and to memory. Each local memory can supply one word of  $k$  bits per cycle. The  $n$  bits of any given node is stored at a different word address. Hence, it takes  $n$  cycles to read all  $n$  bits of each of the  $k$  nodes. The transposer is a two-way shift register array. Actually, there are two transposing buffers so that one can be emptied (written back to memory) and filled (read from memory), while the other is accessed by the collision units. The switch exchanges data with neighboring processors if necessary. At any time, the processor either reads or writes. Since the procedure is deterministic, and the access sequence is data independent, all operations (AG, RD, etc.) are deeply pipelined in order to achieve maximum throughput.

The parameter  $k$  is the number of partitions mapped to a processor. The optimal choice of  $k$  depends on  $n$ , the number of bits per node, the delay through the switch, and the number of I/O pins and area of the VLSI implementation. Some typical numbers we are considering are:  $n = 25$  (1 obstacle bit),  $k = 192$  for a processor with 4 collision units.

#### 4.1 Collision Unit

The properties of a lattice not only govern the propagation phase, but also significantly constrain the collision phase, because the *collision rules* must have the same symmetry as the lattice [4]. How the underlying symmetry group of a lattice gas model can be exploited to derive compact and high performance processing elements to handle collision functions of potentially exponential complexities ( $O(n2^n)$ ) was posed as a major challenge in this area of research (see the Preface of [3]). The FCHC isometric model proposed by Hénon [5] was

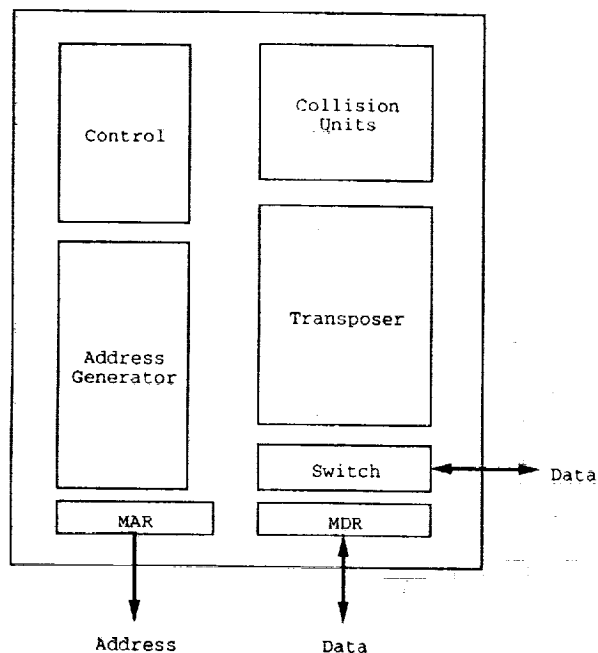


Figure 3: Functional block diagram of the processor

the first real 24-bit three-dimensional model with a detailed specification of an optimized non-deterministic collision function. Therefore, it was chosen as our first case study. A VLSI architecture for the FCHC isometric model has been designed and implemented as an ASIC. We have shown that a 4000 gate chip can replace the equivalent of 4.5 billion bits (rather than 384 million bits due to non-determinism) of a lookup table used to solve this problem. Because the architecture is derived by considering the symmetry properties rather than by brute force logic synthesis, it can be generalized to other classes of lattice gas models. We present the main ideas in this paper. (Please see [8,9] for more details).

Hénon's isometric algorithm [5] shows how the output state of a node is computed as a non-deterministic function of the input state:

1. Compute the momentum of the input state
2. Normalization: Apply the appropriate isometries (symmetry transformations) to the input state and the momentum, so that the momentum is *normalized*.
3. Collision: Choose at random one of the optimal isometries of the class to which the normalized momentum belongs, and apply this isometry.
4. Denormalization: Apply the isometries applied in step 2 in reverse order to obtain the output state.

The application of an isometry to a state is the most frequent and important operation. An efficient implementation of this operation is thus most crucial. Cayley's theorem states that every group is isomorphic to a permutation group, hence it is not too surprising that conditional application of isometries can be implemented as conditional permutations, which in turn map to simple multiplexers. In essence, the algorithm can be viewed as a description of how to generate the right control signals to permute the input state bits.

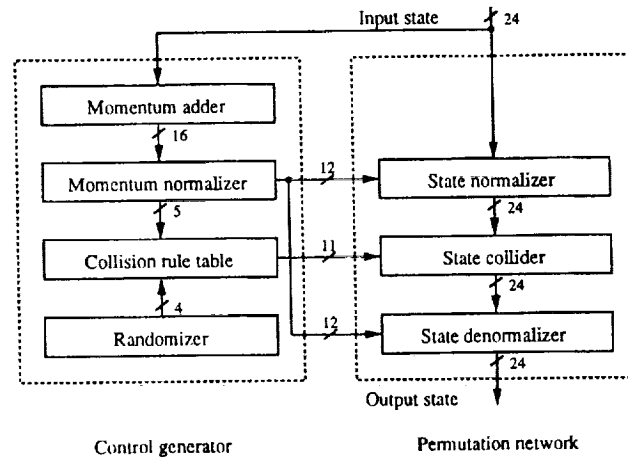


Figure 4: A collision unit

The organization of a collision unit (Figure 4) follows classical lines: the *control path* consisting of a momentum adder, a momentum normalizer, a small collision rule table, and a randomizer; the *data path* is a conditional permutation network composed of a state normalizer, a state collider and a denormalizer (inverse-normalizer). The overall feed-forward character of this unit makes it easy to design a *highly pipelined* version with a proportional increase in throughput.

A CMOS field programmable gate array implementation of the unit with a non-pipelined latency of 460 ns has been completed. A CMOS gate array implementation is estimated to have a non-pipelined latency below 50 ns. A collision unit capable of 20 million node updates per second (MNUPS) or more is clearly feasible. This is comparable to CRAY-2's performance of approximately 30 MNUPS [11].

## 4.2 Address Generator

As large simulation problems require the use of a huge amount of memory, memory chips can easily become the dominant cost factor of the system. Our solution avoids the common but expensive alternative of double buffering the complete memory space, while retaining a high degree of flexibility in the choice of lengths of each dimension of the (simulation) problem space. This is made possible by the *virtual move* addressing mechanism, which exploits the true data dependency of the computation steps involved. Data movement implied by propagation but not by communication requirements can thus be eliminated.

The address generator contains a number of registers and an arithmetic datapath (adder) to generate the complex sequence required.

### 4.2.1 Virtual Move

Although the FCHC models are our major concerns, the mechanisms described below apply to a larger class of models with other possible velocities. The following section is written with general notations so as to be valid for any  $D$ -dimensional space and arbitrary velocity.

The propagation equation (2) seems to suggest that at every time step, state bits of all the nodes have to be moved. However, a closer examination reveals that the equation actually represents an invariant relationship. If we choose to observe in a *frame of reference* moving at the velocity  $\mathbf{v}^i$  with respect to the *rest* frame of the lattice, the particles with velocity  $\mathbf{v}^i$  are obviously stationary! Hence, there is no need to actually *move* the bits in memory, as long as we keep track of the *Galilean transformation*. The coordinate  $\mathbf{x}^i$  of the moving frame is related to the rest coordinate  $\mathbf{x}$  by the transformation:

$$\mathbf{x}^i = \mathbf{x} - \mathbf{v}^i t \quad (4)$$

Suppose the space-time point  $(\mathbf{x}, t)$  corresponds to  $(\mathbf{x}^i, t)$ , then the point  $(\mathbf{x} + \mathbf{v}^i, t + 1)$  corresponds to  $((\mathbf{x} + \mathbf{v}^i) - \mathbf{v}^i(t + 1), t + 1) = (\mathbf{x} - \mathbf{v}^i t, t + 1) = (\mathbf{x}^i, t + 1)$ . Hence, (1) and (2) can be written as

$$\mathbf{b}'(\mathbf{x}^i, t) = \mathcal{C}(\mathbf{b}(\mathbf{x}^i, t)) \quad (5)$$

$$b_i(\mathbf{x}^i, t + 1) = b'_i(\mathbf{x}^i, t) \quad (6)$$

If we interpret  $\mathbf{x}^i$  as the *physical address* used to address memory module  $i$ , then  $\mathbf{x}$  can be treated as the *virtual address*. Equation (6) says that we do not have to move the bits at all in the propagation phase. We refer to this technique as *virtual move*. The cost of implementing virtual move is to have a slightly more complicated address generation scheme. For each virtual address  $\mathbf{x}$ , we need to generate  $n$  physical addresses,  $\mathbf{x}^i$  ( $i = 1, \dots, n$ ) according to (4). However, only one address has to be generated per cycle, if we access one bit plane at a time. Multiple bit planes can be stored in the same memory space by interleaving.

### 4.2.2 Multi-dimensional Modulo Adder

The transformation (4) requires  $D$  modulo subtractions for each  $\mathbf{v}^i$ . How can one proceed to implement the address generators in hardware?

Suppose we have a wrap-around lattice space of dimension  $D$ , implied by the basic type of periodic boundary condition (see section 2). Equation (4) can be written as

$$\begin{aligned} x_\alpha^i &= (x_\alpha - v_\alpha^i t) \bmod n_\alpha \quad (\alpha = 1, \dots, D) \\ &= (x_\alpha + (-v_\alpha^i t \bmod n_\alpha)) \bmod n_\alpha \\ &= (x_\alpha + d_\alpha^i(t)) \bmod n_\alpha \end{aligned} \quad (7)$$

where  $n_\alpha$  is the length of  $x_\alpha$ -dimension, and

$$d_\alpha^i(t) = -v_\alpha^i t \bmod n_\alpha \quad (8)$$

Note that  $d_\alpha^i$  has to be recomputed only once per time step by addition:

$$d_\alpha^i(t+1) = (d_\alpha^i(t) + (-v_\alpha^i \bmod n_\alpha)) \bmod n_\alpha \quad (9)$$

In order to use conventional RAM, we need to map  $\mathbf{x}$  ( $\mathbf{x}^i$ ) to a linear address. We choose the conventional one-to-one mapping

$$A : \{0, 1, \dots, n_1\} \times \{0, 1, \dots, n_2\} \times \{0, 1, \dots, n_D - 1\} \mapsto \{0, 1, \dots, n_1 n_2 \dots n_D - 1\}$$

such that

$$A(\mathbf{x}) = A((x_1, x_2, \dots, x_D)) = x_1 + x_2 n_1 + \dots + x_D \prod_{\alpha=1}^{D-1} n_\alpha \quad (10)$$

Assume that all  $n_\alpha$ 's are powers of 2, such that  $m_\alpha = \log_2 n_\alpha$  and  $\sum_{\alpha=1}^D m_\alpha = m$ . The mapping  $A$  can then be performed trivially by concatenating the binary representations of  $\mathbf{x}$  such that  $x_{\alpha+1}$  is on the left of  $x_\alpha$ . Similarly, we can obtain the linear address of  $d^i$ . Let  $a = A(\mathbf{x})$ , and  $b = A(d^i)$ , and define  $e$  as

$$e_j = \begin{cases} 0 & \text{if } j = \sum_{\kappa=1}^{\alpha} m_\kappa \text{ for some } \alpha \in [0, D-1] \\ 1 & \text{otherwise} \end{cases} \quad (11)$$

The purpose of  $e$  is to mark the boundary bits of dimensions so that *carry-out* from lower dimensions would not be propagated to higher dimensions. The value of  $e$  does not change during a simulation.

We can calculate all  $D$  components of  $\mathbf{x}^i$  according to (7) in one step by using a *multi-dimensional modulo adder*, which takes three  $m$ -bit inputs,  $a$ ,  $b$ , and  $e$ , and computes the sum as  $s = A(\mathbf{x}^i)$ . The adder can be built according to the new definitions of  $p_i$ , propagate, and  $s_i$ , sum:

$$p_i = (a_i \vee b_i) e_i \quad (12)$$

$$s_i = a_i \oplus b_i \oplus c_i e_i \quad (13)$$

and our usual definitions of  $g_i$ , generate, and  $c_i$ , carry:

$$g_i = a_i b_i \quad (14)$$

$$c_0 = 0 \quad (15)$$

$$c_{i+1} = g_i \vee p_i c_i \quad (16)$$

Hence, this modified adder can be implemented in various ways, such as ripple carry adder, carry lookahead adder, or carry select adder, as deemed appropriate for the system requirement and implementation technology.

#### 4.2.3 Example

Let us illustrate the idea by a small example. Suppose we have the 2-D square lattice with  $n = 4$  bits per node, and  $\mathbf{v}^1 = (1, 0)$ ,  $\mathbf{v}^2 = (-1, 0)$ ,  $\mathbf{v}^3 = (0, 1)$ , and  $\mathbf{v}^4 = (0, -1)$ .

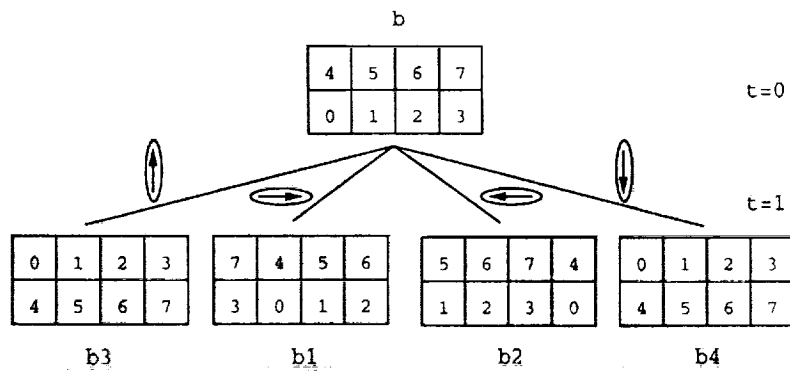


Figure 5: State bit propagation

$\mathbf{x}$	$A(\mathbf{x})$
(0,0)	0
(1,0)	1
(2,0)	2
(3,0)	3
(0,1)	4
(1,1)	5
(2,1)	6
(3,1)	7

Figure 6: Address mapping  $A$ 

$A(\mathbf{x})$	$A(\mathbf{x}^1)$	$A(\mathbf{x}^2)$	$A(\mathbf{x}^3)$	$A(\mathbf{x}^4)$
0	3	1	4	4
1	0	2	5	5
2	1	3	6	6
3	2	0	7	7
4	7	5	0	0
5	4	6	1	1
6	5	7	2	2
7	6	4	3	3

Figure 7: Virtual to real address translation for  $t = 1$



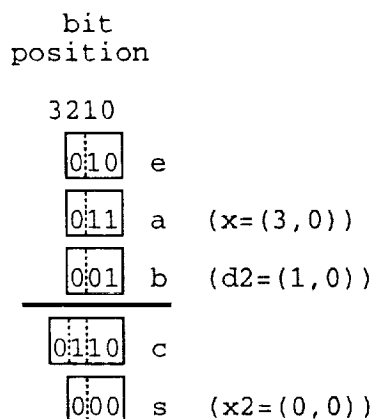


Figure 8: Operation of a multi-dimensional modulo adder.

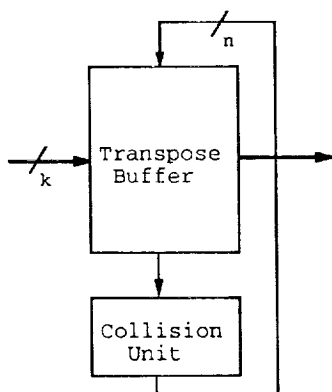
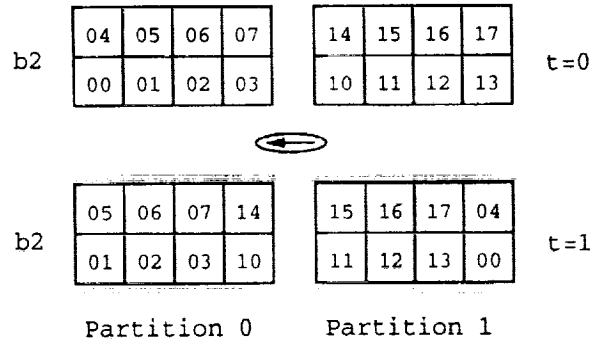


Figure 9: Operation of the transposer

The lattice (problem) space has 8 nodes with  $n_1 = 4$  and  $n_2 = 2$ . In figure 5, each box represents a physical memory location, the linear address mapping of the coordinate of which is given in Figure 6, as calculated by (10). At  $t = 0$ , the virtual address  $\mathbf{x}$  and physical addresses  $\mathbf{x}^i$  ( $i = 1, 2, 3, 4$ ) for any given node are the same. At  $t = 1$ , they are different, as governed by (4) and shown in Figure 7. The numeric label within each box represents a binary value. Suppose we are interested in the *bit plane* of  $b_2$ . The label "3" of the b boxes represents the binary value of  $b_2$  at  $A(\mathbf{x}) = 3$  just after collision at  $t = 0$ . At  $t = 1$  just before collision, the label "3" of the b2 boxes is at a different location, because the bit has been moved to the left by one position, where  $A(\mathbf{x}) = 2$ . However, the move can be avoided if the virtual address 2 is somehow translated into the physical address 3, so that access to  $A(\mathbf{x}) = 2$  becomes access to  $A(\mathbf{x}^2) = 3$ . Figure 8 shows how the translation can be computed by a multi-dimensional modulo adder.

### 4.3 Transposer

It contains 2 transpose buffers, to be filled and emptied alternately. Figure 9 shows the operation of the transposer. It affects memory addressing, data structure to store the array.

Figure 10: Propagation of bits of  $b_2$  across partitions

During the  $i$ -th cycle, the  $i$ -th bits of the  $k$  words are read and shifted into a transpose buffer. At the end of the  $n$  cycles, the  $n$  bits of one node are shifted out per cycle. A collision unit takes the bits as input, and the output is written back to the buffer. It acts as a circular shift register. With multiple ( $u$ ) collision units, multiple updates (collisions) can be executed in parallel per cycle. This update continues for  $k/u$  cycles until all  $k$  nodes are processed. They are then shifted out bit-by-bit to memory. While one buffer is busy acting as an  $n$ -wide circular shift register to serve the collision units, the other can be emptied and refilled just in time to take the turn, if  $k$  is chosen appropriately.

#### 4.4 Switch

Updating a node at the border of a partition requires reading values from one or more adjacent partitions. We need to know when to select data bits from which partitions.

According to (2), we know where the neighboring nodes are in the problem space:

$$b_i(\mathbf{x}, t+1) = b'_i(\mathbf{x} - \mathbf{v}^i, t) \quad (17)$$

Let us define three coordinate systems, namely, the *global*, *partition*, and *local* coordinates such that they satisfy the following relationship:

$$\mathbf{x}^G = \mathbf{P}\mathbf{x}^P + \mathbf{x}^L \quad (18)$$

where  $\mathbf{P}$  is a diagonal matrix with  $p_{\alpha\alpha} = n_\alpha$ , and the following conditions are satisfied:

$$0 \leq x_\alpha^L < n_\alpha, \quad 0 \leq x_\alpha^P < p_\alpha, \quad 0 \leq x_\alpha^G < n_\alpha p_\alpha \quad (19)$$

Alternatively, we can write for any  $\alpha$

$$x_\alpha^G \bmod n_\alpha p_\alpha = n_\alpha (x_\alpha^P \bmod p_\alpha) + x_\alpha^L \bmod n_\alpha \quad (20)$$

Then we can show that for any  $\alpha$ ,

$$(x_\alpha^G - v_\alpha^i) \bmod n_\alpha p_\alpha = n_\alpha ((x_\alpha^P + \text{bound}(0, x_\alpha^L - v_\alpha^i, n_\alpha)) \bmod p_\alpha) + (x_\alpha^L - v_\alpha^i) \bmod n_\alpha \quad (21)$$

where bound is defined as

$$\text{bound}(L, k, U) = \begin{cases} -1 & \text{if } k < L \\ 0 & \text{if } L \leq k < U \\ 1 & \text{if } k \geq U \end{cases} \quad (22)$$

For any given  $\mathbf{v}^i$ ,  $\text{bound}(0, x_\alpha^L - v_\alpha^i, n_\alpha)$  is either non-negative or non-positive. Hence, it is only necessary to distinguish whether the value is zero or non-zero.

In equation (21), the partition coordinate determines the source partition, and the local coordinate determines the bit within a partition. Since the machine is synchronous, at any one time, all  $x_\alpha^L$  have the same value for all partitions. This exactly matches the requirement implied by (21).

Figure 10 shows an example. It shows the data distribution at  $t = 1$  for  $b_2$  for the same example shown in Figure 5. The first digit of a label represents the partition coordinate mapping, while the second one represents the local coordinate mapping.

The function bound can be computed as a *carry-out* of the multi-dimensional modulo adder. Let  $a = A(\mathbf{x}^L)$ ,  $d_\alpha^i = -v_\alpha^i$ , and  $e$  as defined before, as the inputs to a multi-dimensional modulo adder, then for each  $\alpha$ ,  $|\text{bound}(0, x_\alpha^L - v_\alpha^i, n_\alpha)|$  is exactly the carry-out bit which is to be blocked so that it will not flow across the dimension boundary. In Figure 8 the carry bit  $c_2 = 1$  indicates that the local virtual address 3 of a partition maps to the local physical address 0 of its adjacent partition, as shown in Figure 10.

## 5 Summary

We have outlined a number of unique architectural features of a very high performance pipelined array processor dedicated to lattice gas simulation. The architecture is truly scalable in the sense that it achieves linear speedup for both fixed and increasing problem sizes with more processors. It is necessary and possible to take advantage of the special properties of the application to design application-specific computers that are a thousand times more powerful than existing supercomputers.

The driving limitation of ALGE is memory bandwidth. This situation becomes more severe as the processing elements run faster and the clock cycle gets shorter. This may be an ideal project for the use of high density mounting and packaging technology such as multiple chip modules.

Current work is focusing on resolving finer issues of design and implementation with the goal of building a prototype system.

The promise of powerful VLSI processors for digital wind tunnels opens up the potential for desk-top and onboard applications.

## References

- [1] Andre Clouqueur and Dominique d'Humières. R.A.P., A Family of Cellular Automaton Machines for Fluid Dynamics. *Helvetica Physica Acta*, 62:525-541, 1989.

- [2] K. Diemer, K. Hunt, S. Chen, T. Shimomura, and G. Doolen. Density and velocity dependence of reynolds numbers for several lattice gas models. *Lattice Gas Methods for Partial Differential Equations*, pages 137–177, 1990.
- [3] Gary D. Doolen, editor. *Lattice Gas Methods for Partial Differential Equations*, volume IV of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, 1990.
- [4] Uriel Frisch, Dominique d’Humières, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau, and Jean-Pierre Rivet. Lattice Gas Hydrodynamics in Two and Three Dimensions. *Complex Systems*, 1(4):649–707, 1987.
- [5] Michel Hénon. Isometric Collision Rules for the Four-Dimensional FCHC Lattice Gas. *Complex Systems*, 1(3):475–494, June 1987.
- [6] Steven D. Kugelmass. *Architectures for Two-Dimensional Lattice Computations with Linear Speedup*. PhD thesis, Princeton University, June 1988.
- [7] Fung F. Lee and Michael J. Flynn. Architectural Mechanisms to Support Three-Dimensional Lattice Gas Simulations. *Third Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 115–122, July 1991.
- [8] Fung F. Lee, Michael J. Flynn, and Martin Morf. A VLSI Architecture for the FCHC Isometric Lattice Gas Model. Technical Report CSL-TR-90-426, Computer Systems Laboratory, Stanford University, April 1990.
- [9] Fung F. Lee, Michael J. Flynn, and Martin Morf. Design of Compact High Performance Processing Elements for the FCHC Lattice Gas Models. *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, March 1991.
- [10] Norman Margolus and Tommaso Toffoli. Cellular Automata Machines. *Lattice Gas Methods for Partial Differential Equations*, pages 219–249, 1990.
- [11] Jean-Pierre Rivet, Michel Hénon, Uriel Frisch, and Dominique d’Humières. Simulating Fully Three-Dimensional External Flow by Lattice Gas Methods. *Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics*, pages 276–285, September 1988.
- [12] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines – A New Environment for Modeling*. MIT Press, 1987.

## Verification of VLSI Designs

P. J. Windley  
Department of Computer Science  
University of Idaho  
Moscow, ID 83843  
m 208.885.6501

**Abstract:** In this paper we explore the specification and verification of VLSI designs. The paper focuses on abstract specification and verification of functionality using mathematical logic as opposed to low-level boolean equivalence verification such as that done using BDDs and Model Checking. Specification and verification, sometimes called *formal methods*, is one tool for increasing computer dependability in the face of an exponentially increasing testing effort.

### 1 Introduction

Reliable computer systems are becoming increasingly difficult to engineer. The successes of IC fabrication technology have put VLSI engineers in the position of building dependable computers that are orders of magnitude more complex than the largest computers of even a decade ago. With even larger numbers of transistors promised in the near term, research is being done to make the reliable engineering of complex VLSI designs practical.

There are two complimentary approaches to computer reliability: fault tolerance and fault exclusion. The former is most useful in handling dynamic faults occurring during system operation due to component failure or other unexpected events. The latter is a static process intended to remove errors in design and implementation before the computer system is in service.

Testing and simulation are well-known fault exclusion techniques. Testing and simulation are used extensively in the design, implementation, and manufacturing of computer systems. The problem is that testing and simulation can never exhaustively cover every possible situation that the circuit might encounter. Pygott [13] states

"A comparatively simple 8-bit microprocessor such as the Z80 has 208 internal memory elements and 13 input signals, meaning that the circuit is capable of  $2^{221}$  different state transitions. Even if a transition could be simulated every microsecond, it would take  $10^{53}$  years to examine all the possible changes (this is far larger than the age of the universe)."

Clearly, only a tiny fraction of the possible state transitions can be tested. This situation has led to VLSI devices going to market with design faults which were not caught in testing.

One possible answer to the inadequacies of testing and simulation is hardware synthesis from high-level circuit descriptions written in an appropriate hardware description language (HDL) such as VHDL [7]. Synthesis from an HDL description certainly has much promise. Textual descriptions are easy to store, manipulate, and process. Also, synthesis

### 7.3.2

tools are likely to be reliable since the social process of hundreds of users using a synthesis program tends to exorcise any latent bugs.

Unfortunately, synthesis of VHDL circuit descriptions is not sufficient for dependable computing. As a case in point, consider that high-level programming languages have been in use for 20 years and programs still contain numerous errors. There are a number of reasons why this is so:

1. HDLs are generally verbose making them hard to read.
2. HDL constructs are not usually amenable to formal analysis. Thus it is nearly impossible to show that a particular description has desired properties.
3. Constructs that can be synthesized are frequently not abstract enough to be of use as system specifications.
4. Contrary to what the marketers of synthesis systems would have one believe, circuit descriptions outside of a small subset of a HDL cannot be synthesized. This is particularly true of abstract descriptions. One need not search further than a multiplier to find an example of this.

Because of these limitations in testing, simulation, and synthesis, much effort is being expended in the formal specification and verification of hardware. Formal methods offer hope of overcoming some of these shortcomings because they are based on logic and can thus take advantage of the decades of mathematical research on using logical analysis.

1. Logical circuit descriptions are often more concise than conventional HDL descriptions.
2. Numerous formalisms can be embedded in logic. This allows the circuit specifier to use the most appropriate formalism for the job [10].
3. One can prove properties about logic descriptions directly using a proof system such as predicate calculus. This can be very effective for establishing that a specification meets its requirements [17].
4. Analysis can be applied to the specification and less-abstract structural circuit descriptions to show functional correctness [2,9,16].
5. Logic provides behavioral, structural, data, and temporal abstraction mechanisms for reducing the complexity of the description [12].

For these and other reasons, we believe that formal methods can play an important part in increasing the reliability of computer systems.

Note that we are not suggesting that formal methods *replace* testing, simulation, and synthesis, but rather that they *complement* these techniques. Figure 1 shows an idealized ASIC design process (adapted from [11]). The *RTL circuit description* is written in an

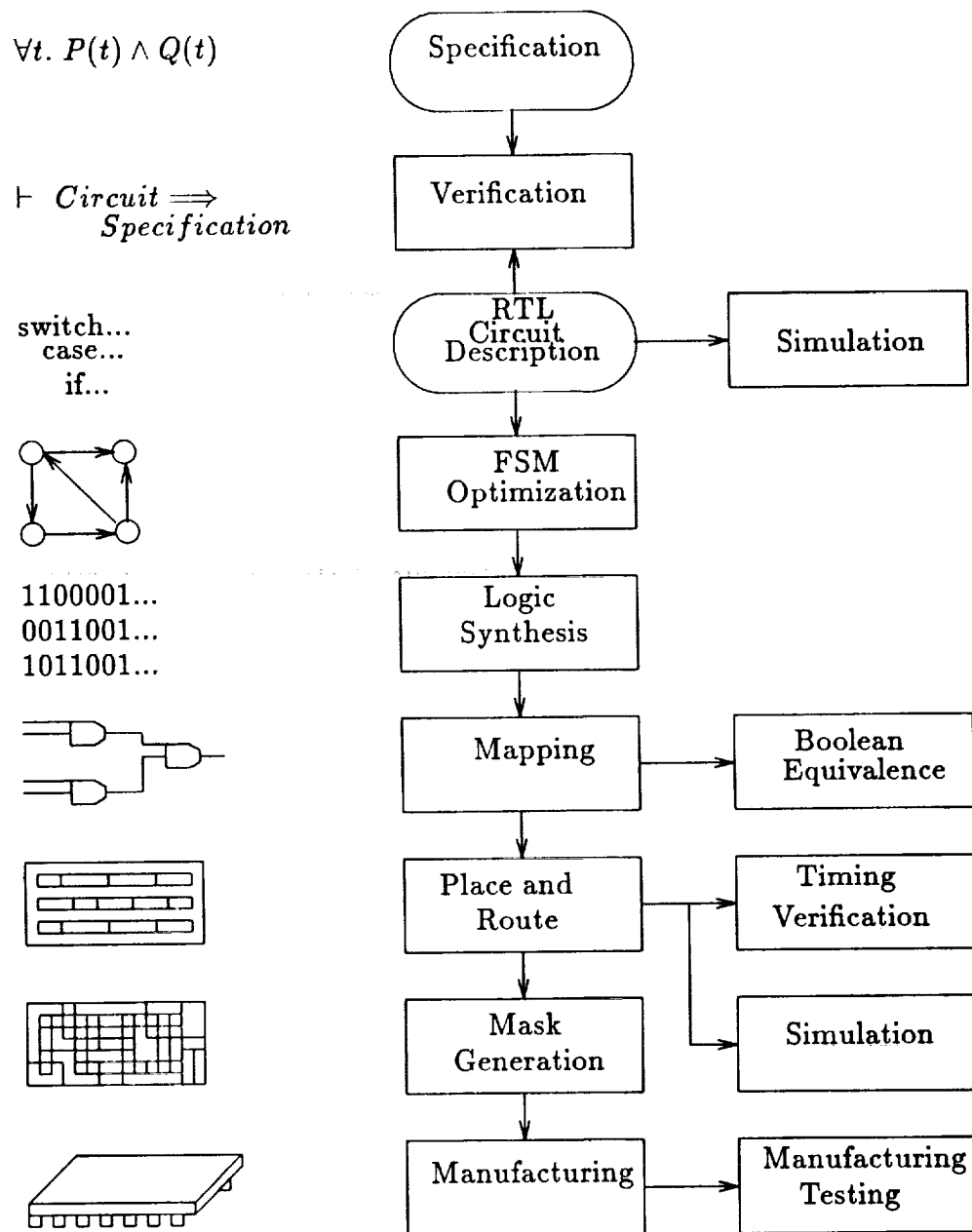


Figure 1: The ASIC Design Process.

appropriate HDL and subjected to synthesis and simulation. The *specification* is a more abstract declarative description which is subject to formal analysis.

Having described the benefits of formal methods, there are a number of directions that we could take. We could, for example, focus on how formal methods interacts with conventional CAD tools or discuss the pros and cons of the design process in Figure 1. Instead, we will show how logic can be used to specify circuits and how proof functions as a mathematical analysis tool for reasoning about those specifications. We do not attempt to give a complete survey of the field, but rather focus on a demonstration of techniques.

## 2 Using Logic to Specify Hardware.

A circuit is a collection of devices composed by interconnection. Each of these devices has ports which are used for input, output, or both. The behavior of a device can be expressed in terms of its ports. Each of the devices in a circuit can, in turn, be viewed as a composition of still other devices. This hierarchy of devices eventually leads to the devices that the designer considers primitive. The smallest devices we will deal with in this paper are logic gates and indeed, in many cases, we will stop much higher than even gates.

Clocksin describes several ways to specify circuit structure [3]:

- We can use imperative declarations of the circuit structure (this is referred to as the extensional method).
- We can use functions to describe the output in terms of the input.
- We can use predicates in a quantified logic to relate the ports of a device using behavioral or structural constraints.

Each of these methods has advantages and disadvantages. The extensional method has the advantage of being familiar to designers since it resembles imperative languages such as Pascal that most designers have used. Most modern hardware descriptions languages (e.g. VHDL) use the extensional method. The largest disadvantage of the extensional method is that it is difficult to treat formally, just as imperative programming languages are hard to treat formally.

The functional model is widely used; Hunt's specification of the FM8501 microprocessor, for example, is functional [6]. To specify the behavior of sequential circuits functionally, the specification language must support recursion. Hunt uses recursion to describe the sequential operation of his CPU.

In the functional model, circuit interconnection is given by the syntactic structure of function application. This can cause several problems:

- Describing circuits with bi-directional ports is difficult since functional specifications differentiate between input and output syntactically.
- The purpose of a structural specification is to show how components are connected together. Since the only means of expressing connection is function application, even returning a tuple is insufficient for describing circuits with more than one output.



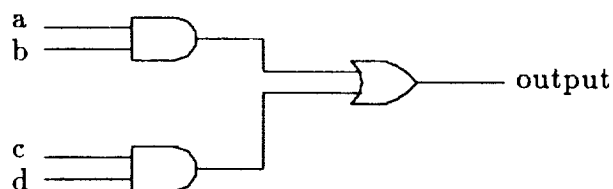


Figure 2: Implementation of a simple circuit, D

- Sequential circuits feedback on themselves. Recursion is the best alternative; but that can be inadequate for circuits with multiple feedback paths.

The predicate method is a widely used specification technique [5] and is the one we will demonstrate in this paper. A disadvantage of the predicate method is that designers are likely to find it the most unfamiliar of the three and thus difficult to use. In addition, to use the predicate method, the logic must support existential quantification, either explicitly or implicitly. (Prolog is an example of a language with implicit existential quantification.) The predicate method does, however lend itself to a wide variety of circuit types, including those with multiple outputs and bi-directional ports.

## 2.1 Specifying Circuits with Predicates.

As an example of the predicate model, we will specify the behavior and structure of a very simple circuit we call D. The predicate that specifies the behavior of the circuit can be given by the following logic definition:

$$\vdash_{def} D(a,b,c,d,out) = out = (a \wedge b) \vee (c \wedge d)$$

Notice that the inputs and outputs are all included in the arguments and the behavior is expressed as a constraint among the outputs and the inputs.

One possible implementation for D is shown in Figure 2. As was mentioned earlier, each device can be thought of as representing a constraint on its inputs and outputs. For example, the top And gate constrains *a*, *b*, and *p* in a manner consistent with the behavior of the device.

$$\vdash_{def} And(a, b, p) = (p = a \wedge b)$$

To get the constraint represented by the entire device, we can compose the individual constraints using conjunction.

$$And(a, b, p) \wedge And(c, d, q) \wedge Or(p, q, out)$$

This expression constrains the values not only on the ports of the device, *a*, *b*, *c*, *d*, and *out*, but also on the internal lines *p* and *q*. We normally wish to regard such a device as a "blackbox" and consequently are only interested in the values of the external lines. We can hide the internal lines using existentially quantified variables and define a predicate *D\_imp* that represents the structure of the circuit.

### 7.3.6

$$\vdash_{def} D\_imp(a, b, c, d, out) = \\ \exists p\ q. \quad And(a, b, p) \wedge And(c, d, q) \wedge Or(p, q, out)$$

While this formula looks confusing at first, we should note that this level of specification can be produced automatically from netlists or traditional HDL models.

For comparison, the following specification describes the same circuit using functions:

$$\vdash_{def} D(a,b,c,d) = \quad Or(And(a,b),And(c,d))$$

The outputs are not mentioned explicitly; the result of the function is taken to be the output of the circuit.

Similarly, we can write an extensional specification of the circuit in a hardware description language such as VHDL [1]:

```
Entity D_imp is
  port(a, b, c, d :in Bit; outp :out Bit);
end D_imp;

architecture Structure of D_imp is
  component ANDGate port(i1,i2:in Bit; outp :out Bit);
  component ORGate port(i1,i2:in Bit; outp :out Bit);
  signal p, q: Bit

  G1: ANDGate port map (a, b, p);
  G2: ANDGate port map (c, d, q);
  G3: ORGate port map (p, q, outp);

end Structure;
```

The difference between this specification and the predicate model of the circuit structure is largely superficial. The primary difference is the abundance of keywords in the extensional specification. The biggest impediment to using specification languages such as VHDL is that they sometimes lack a clear semantics. This problem can be overcome by defining a semantics of the specification language in the object language of a verification tool such as HOL. Van Tassel has done just that using VHDL and HOL in [14,15].

## 2.2 Specifying Sequential Behavior.

The last section specified a simple combinatorial circuit. We specify the behavior of sequential circuits in higher-order logic using an explicit representation of time.

For example, we can specify the behavior of a simple latch as follows:

$$\vdash_{def} latch\ in\ out\ set = \forall t. \ out\ (t+1) = set\ t \rightarrow in\ t \mid out\ t$$

In the specification, *in*, *out*, and *set* are functions of time. The value of a signal at time *t* is returned when the function representing the signal is applied to *t*. The specification says that the value of *out* at time *t* + 1 gets the value of the input port, *in*, at time *t* if

the set line is high and remains unchanged otherwise. Universal quantification over time is used in defining the predicate.

We can also use existential quantification to describe temporal operators. For example, suppose that we wish to define a predicate that says that a signal will *eventually* go high. The following is a definition of an EVENTUALLY operator:

$$\vdash_{def} \text{EVENTUALLY } d \ t1 = \exists \ t2. \ t2 > t1 \wedge d \ t2$$

When applied to the signal  $d$ , and the current time,  $t1$ , the predicate states that there exists a time,  $t2$ , in the future when the signal  $d$  will be true. The use of existential quantification over time is also used to specify the behavior of asynchronous interconnections between devices. Joyce [9] has shown how temporal logic can be embedded in higher-order logic.

## 2.3 Behavioral Abstraction and Specification.

There are many ways of specifying the same circuit. For example, in specifying a two input binary decoder, one might write:

$$\begin{aligned} \vdash_{def} \text{decoder\_spec } s0 \ s1 \ o0 \ o1 \ o2 \ o3 = \\ & (o0 = (s1 \rightarrow (s0 \rightarrow F \mid F) \mid (s0 \rightarrow F \mid T))) \wedge \\ & (o1 = (s1 \rightarrow (s0 \rightarrow F \mid F) \mid (s0 \rightarrow T \mid F))) \wedge \\ & (o2 = (s1 \rightarrow (s0 \rightarrow F \mid T) \mid (s0 \rightarrow F \mid F))) \wedge \\ & (o3 = (s1 \rightarrow (s0 \rightarrow T \mid F) \mid (s0 \rightarrow F \mid F))) \end{aligned}$$

While this specification is correct, its meaning is not very clear.

Here is another specification for the same behavior:

$$\begin{aligned} \vdash_{def} \text{decoder\_spec } s0 \ s1 \ o0 \ o1 \ o2 \ o3 = \\ & (o0 = \neg s1 \wedge \neg s0) \wedge \\ & (o1 = \neg s1 \wedge s0) \wedge \\ & (o2 = s1 \wedge \neg s0) \wedge \\ & (o3 = s1 \wedge s0) \end{aligned}$$

This specification closely models one possible implementation for the circuit; consequently, using it as the behavioral specification would make the verification easier, but would not tell us much about the abstract behavior of the decoder.

The next specification is more abstract and says more about the behavior of the decoder:

$$\begin{aligned} \vdash_{def} \text{decoder\_spec } s0 \ s1 \ o0 \ o1 \ o2 \ o3 = \\ & (o0 \leftrightarrow ((s1, s0) = (F, F))) \wedge \\ & (o1 \leftrightarrow ((s1, s0) = (F, T))) \wedge \\ & (o2 \leftrightarrow ((s1, s0) = (T, F))) \wedge \\ & (o3 \leftrightarrow ((s1, s0) = (T, T))) \end{aligned}$$

This specification clearly shows the binary numbers being represented by the inputs. Moreover, the specification does not suggest any particular implementation. In general, the more abstract a specification, the easier it is to understand, but more difficult it is to verify.

We can make the above specification even more abstract by defining a function, *pairval*, that converts boolean pairs into numbers and then writing the specification as follows.

```

decoder_spec s0 s1 o0 o1 o2 o3 =
  let n = pairval(s1,s0) in
    (o0 ↔ (n = 0)) ∧
    (o1 ↔ (n = 1)) ∧
    (o2 ↔ (n = 2)) ∧
    (o3 ↔ (n = 3))

```

This specification can be readily generalized to have  $n$  inputs and  $2^n$  outputs.

## 2.4 Specifying a Microprocessor

So far, the circuits we have described have been simple, for expository purposes. One should not assume that all specifications must be of small devices. Indeed, logic is most useful when used on large, abstract specifications. To demonstrate the use of formal specification on a larger example, we will present the specification of a small microprocessor called Tamarack.

There have been numerous efforts to verify microprocessors [4,8,6]. Most of these have used the same implicit behavioral model. In general, the model uses a state transition system to describe the microprocessor. A microprocessor specification has four important parts:

1. A representation of the state, **S**. This representation varies depending on the verification system being used.
2. A set of state transition functions, **J**, denoting the behavior of the individual instructions of the microprocessor. Each of these functions takes the state defined in step (1) as an argument and returns the state updated in some meaningful way.
3. A selection function, **N**, that selects a function from the set **J** according to the current state.
4. A predicate, **I**, relating the state at time  $t + 1$  to the state at time  $t$  by means of **J** and **N**.

In some cases, the individual state transition functions, **J**, and the selection function, **N**, are combined to form one large state transition function.

To make all of this more concrete consider the top-level specification of Tamarack presented by Joyce in [9].

```

⊢def TamarackBeh (ireq, mem, pc, acc, rtn, iack) =
  ∀t:time.
    (mem (t+1), pc (t+1), acc (t+1), rtn (t+1), iack (t+1)) =
      NextState (ireq t, mem t, pc t, acc t, rtn t, iack t)

```

The top-level specification relates the state of the assembly language level registers at time  $t + 1$  to their state at time  $t$  using the function **NextState**. The level of abstraction in the

top-level specification is roughly that found in an assembly language reference manual. The difference is that the formal specification is less ambiguous and more complete.

The next state function chooses among the many individual instructions according to a selection criteria which describes, in an abstract way, instruction decoding:

```

 $\vdash_{def}$  NextState (ireq, mem, pc, acc, rtn, iack) =
  let opcval = OpcVal (mem,pc) in
  ((ireq  $\wedge$   $\neg$ iack)  $\rightarrow$  IRQ_SEM (mem,pc,acc,rtn,iack) |
   (opcval = JZR_OPC)  $\rightarrow$  JZR_SEM (mem,pc,acc,rtn,iack) |
   (opcval = JMP_OPC)  $\rightarrow$  JMP_SEM (mem,pc,acc,rtn,iack) |
   (opcval = ADD_OPC)  $\rightarrow$  ADD_SEM (mem,pc,acc,rtn,iack) |
   (opcval = SUB_OPC)  $\rightarrow$  SUB_SEM (mem,pc,acc,rtn,iack) |
   (opcval = LDA_OPC)  $\rightarrow$  LDA_SEM (mem,pc,acc,rtn,iack) |
   (opcval = STA_OPC)  $\rightarrow$  STA_SEM (mem,pc,acc,rtn,iack) |
   (opcval = RFI_OPC)  $\rightarrow$  RFI_SEM (mem,pc,acc,rtn,iack) |
   NOP_SEM (mem,pc,acc,rtn,iack))

```

Each of the instructions available to the programmer as well as actions that take place on instruction boundaries such as interrupts are defined using a function on the state and environment variables that returns a new state updated as appropriate for the instruction being specified. We use the ADD instruction as an example:

```

 $\vdash_{def}$  ADD_SEM (mem:*memory,pc:*wordn,acc:*wordn,rtn:*wordn,iack:bool) =
  let inst = fetch (mem,(address pc)) in
  let operand = fetch (mem,(address inst)) in
  (mem, inc pc, add(acc,operand), rtn, iack)

```

This instruction increments the program counter and stores the result of adding the accumulator to the contents of memory pointed to by the current instruction in the accumulator. No other state changes occur.

There are at least three kinds of abstraction taking place between the register transfer level (RTL) description of Tamarack and the top-level specification given above.

1. **Behavioral Abstraction** — The RTL description of Tamarack is a structural model that says how the major blocks are connected. The top-level specification says nothing about the structure of the microprocessors, but rather describes the required behavior.
2. **Data Abstraction** — The RTL description contains registers that are not of interest in the top-level specification. A good example of these types of registers is the instruction register which is vital to the correct functioning of the microprocessor, but is not considered in the top-level specification.
3. **Temporal Abstraction** — Events at the RTL level happen at a much finer time granularity than events at the top-level. Events at the top-level are measured on a time-scale that coincides with the execution of macro-level instructions. Events at the RTL level are measured by the sub-cycle clock. Many RTL level events must take place to cause one top-level event to happen.

### 3 Using Proof to Analyze Specifications

Proof can be used in at least two ways to analyze specifications. The first method asks the question *Is my specification correct?* The second method asks the question *Does my implementation meet the specification?*

#### 3.1 Design Verification

Determining whether or not a specification is correct is not a question that can be subjected to exhaustive mathematical analysis since the design is an intellectual artifact, not a mathematical one. We can, however, determine whether a specification meets its requirements to the extent that those requirements can be formulated in logic.

An example of this is the verification of two important properties of the supervisory mode of a microprocessor called *AVM-1* [17]. *AVM-1* has a supervisory mode that is controlled by the supervisory mode bit in a register called the program status word (PSW). When the processor is in supervisory mode, certain registers in the register file (which does not include the PSW) become writable. Otherwise they can only be read.

One of the design requirements can be stated informally as follows:

**Property 1 (Integrity of Privileged Registers)** *If the CPU is not in supervisory mode and the next instruction is not an external or user-generated interrupt, then every privileged register remains unchanged.*

The integrity of the privileged registers is only important at the assembly language programmer's level of the CPU. We do not care if the registers change on a finer time scale so long as they remain the same when viewed by the outside world.

The formalization of this requirement is not difficult. The following expression captures the essence of the problem:

$$\forall n . (IS\_SUP\_REG\ n) \Rightarrow \\ (EL\ n\ (macro\_reg\ (t+1)) = \\ (EL\ n\ (macro\_reg\ t))))$$

The expression states that the register file (represented by a list) is the same for every supervisory mode register at time  $t + 1$  as it was at time  $t$ .<sup>1</sup>

The basic requirement, stated above, must follow from the definition of the top-level of *AVM-1* (*AVM\_Beh*) and is subject to the following conditions:

1. The CPU is not currently in supervisory mode (expressed as  $\neg get\_sm\ (psw\ t)$ ).
2. The next instruction is not an internal or external interrupt (expressed in the specification as  $\neg (Opcode\ \dots = INT\_OPCODE)$  and  $\neg (Opcode\ \dots = EINT\_OPCODE)$ ).

---

<sup>1</sup>EL selects the  $n^{th}$  member of a list.

```

⊢ AVH_Beh
  (λ t. (reg_list t,psw t,pc t,mem t,ivec t))
  (λ t. (ireq_e t)) ⇒
  (∀ t.
    ¬get_sm (psw t) ∧
    ¬(Opcode (reg_list t,psw t,pc t,mem t,ivec t)
      (ireq_e t) = INT_OPCODE) ∧
    ¬(Opcode (reg_list t,psw t,pc t,mem t,ivec t)
      (ireq_e t) = EINT_OPCODE) ⇒
    (∀n. IS_SUP_REG n ⇒
      (EL n(reg_list(t + 1)) = EL n(reg_list t))))

```

This theorem is not difficult to establish and, when combined with a correctness proof (see Section 3.2), gives confidence that the supervisory mode works as it should.

### 3.2 Functional Verification

A second, and complimentary, use of proof is in showing that our specification is correctly implemented by the structure that we have chosen for the RTL model.

A simple example is given by the circuit D specified in Section 2.1. To show that the implementation (represented by D\_imp) meets its specification (represented by D), we prove the following theorem:

```

⊢ ∀ a b c d out . D_imp(a,b,c,d,out) ⇒ D(a,b,c,d,out)

```

This theorem could be proven using any number of techniques. Indeed, while it is a simple example, it has little to do with the kinds of proofs of correctness that occur most frequently or that are the most interesting.

A more interesting example is given in the proof of correctness of Tamarack [9] since the proof involves behavioral, data, and temporal abstraction. We have already seen the specification of the top-level of Tamarack (see Section 2.4). The RTL model is a fairly large, but conventional description of the large grain structure of the microprocessor.

In order to understand the correctness theorem, we must describe the temporal abstraction that takes place between the RTL model and the top-level behavioral description. As we have already mentioned, different levels in the specification have different views of time. We use temporal abstraction to produce a function that maps time at one level to time at another. Figure 3 shows a temporal abstraction function  $\mathcal{F}$ . The circles represent clock ticks. Note that the number of clock ticks required at the bottom-level to produce one clock tick at the top-level is irregular.

The predicate,  $\mathcal{G}$ , is true whenever there is a valid abstraction from the lower level to the upper level. We can define a generic temporal abstraction function in terms of  $\mathcal{G}$ . In a microprocessor specification,  $\mathcal{G}$  is usually a predicate indicating when the lower level machine is at the beginning of its cycle—a condition that is easy to test.

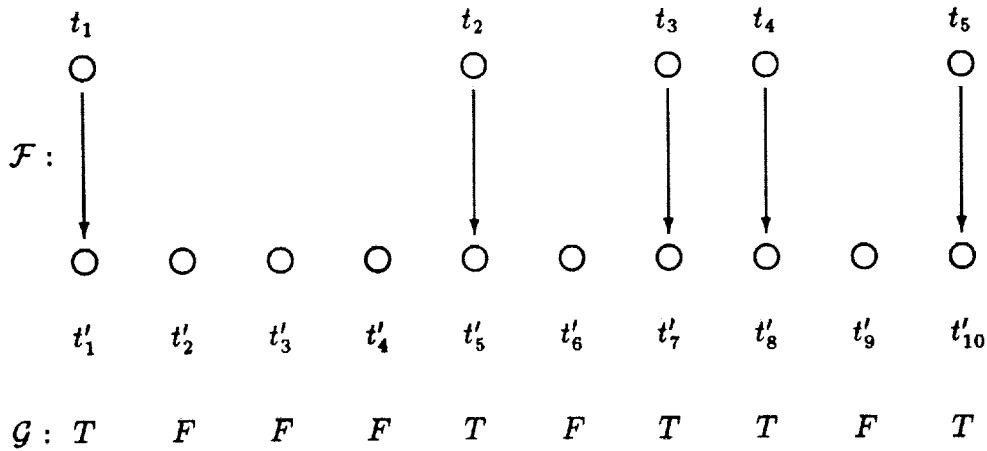


Figure 3: The function  $\mathcal{F}$ , which maps time at one level to another, can be defined in terms of a predicate,  $\mathcal{G}$ , which is true only when the mapping occurs.

We will use a function `TimeOf` as our temporal abstraction function. The function is defined recursively so that `(TimeOf g 0)` is the first time that the predicate  $g$  is true and `(TimeOf g (n+1))` is the next time after time  $n$  when  $g$  is true. We will not develop the details of the temporal abstraction function here, but refer the interested reader to [9].

The final correctness theorem for Tamarack states that the behavioral model (defined by `TamarackBeh`) follows from a system (`AsynSystem`) composed of the RTL model and an asynchronous memory subsystem.

```

⊢ AsynSystem (idreq,mpc,mar,pc,acc,ir,rtn,arg,buf,idack,dack,mem) ∧
  ((val4 o mpc) 0 = 0)
⇒
  let f = TimeOf (((val4 rep) o mpc) Eq 0) and (not dack)) in
  TamarackBeh (idreq o f,mem o f,pc o f,acc o f,rtn o f,idack o f)

```

The function  $f$  is the function  $\mathcal{F}$  of Figure 3. We also have a reset condition that requires that the value of the microprogram counter, `mpc`, be 0 at time 0.

Presenting the proof of the correctness theorem for Tamarack is beyond the scope of this paper. The proof is actually quite straightforward in most cases, involving standard proof techniques such as substitution, case analysis, and induction. Indeed, much of the difficulty is caused by the size of the proof effort rather than the puzzling nature of the theorems. Tamarack is, of course, far from being the largest device with a verified correctness. Recent research has developed techniques for managing much of the complexity of proofs of this sort [16]. The techniques are demonstrated in the proof of correctness of *AVM-1*.

One should not, of course, accept that the microprocessor is correct simply because there is a theorem. The idea is that proof constitutes engineering analysis and like an engineering analysis, must be documented and subject to review. What we have presented here is not, of course, an engineering analysis.



## 4 Conclusions

This paper has shown how logic can be used to specify and analyze hardware designs. The use of formal methods has a number of advantages.

- Specifications give a clear and precise statement of the intended behavior of a design.
- Specifications can be analyzed to determine whether or not they meet the requirements of the design.
- Functional correctness can be demonstrated through analysis rather than testing.
- Assumptions are made explicit.

We do not suggest that formal methods replace conventional engineering practices, but augment them. Work is continuing to bring tools based on formal methods into the designers toolbox:

- We are developing new high-level models of common hardware devices which guide the specification and verification of those devices.
- We are writing translators between hardware description languages used by conventional CAD tools and verification tools.
- We are doing case studies to serve as examples of specification and verification.

These efforts, and similar efforts at other institutions promise to make formal methods tractable for large-scale use in VLSI design.

## References

- [1] J. R. Armstrong. *Chip-Level Modeling with VHDL*. Prentice Hall, 1989.
- [2] A. Camilleri, M. Gordon, and T. Melham. Hardware verification using higher order logic. In D. Borriane, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*. Elsevier Scientific Publishers, 1987.
- [3] W. F. Clocksin. Logic programming and digital circuit analysis. *The Journal of Logic Programming*, 4:59-82, 1987.
- [4] A. Cohn. Correctness properties of the VIPER block model: The second level. Technical Report 134, University of Cambridge Computer Laboratory, May 1988.
- [5] M. J. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153-177. Elsevier Scientific Publishers, 1986.

- [6] W. A. Hunt. The mechanical verification of a microprocessor design. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*. Elsevier Scientific Publishers, 1987.
- [7] IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual*, 1987.
- [8] J. J. Joyce. Formal verification and implementation of a microprocessor. In G. Birtwhistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*. Kluwer Academic Press, 1988.
- [9] J. J. Joyce. *Multi-Level Verification of Microprocessor-Based Systems*. PhD thesis, Cambridge University, December 1989.
- [10] J. J. Joyce. More reasons why higher-order logic is a good formalism for specifying and verifying hardware. In *Proceedings of the ACM/SIGDA International Workshop in Formal Methods in VLSI Design*, January 1991.
- [11] K. Keutzer. Panel discussion: Model checking, theorem proving, and CAD. In *ACM/SIGDA International Workshop in Formal Methods in VLSI Design*, January 1991.
- [12] T. Melham. Abstraction mechanisms for hardware verification. In G. Birtwhistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*. Kluwer Academic Publishers, 1988.
- [13] C. Pygott. Noden.HDL: an engineering approach to hardware verification. In G. Milne, editor, *The fusion of Hardware Design and Verification*. Elsevier Science Publ. B.V.IFIP, 1988.
- [14] J. P. V. Tassel. The semantics of VHDL with VAL and HOL: Towards practical verification tools. Master's thesis, Department of Computer Science and Engineering, Wright State University, 1989.
- [15] J. P. V. Tassel and D. Hemmendinger. Toward formal verification of VHDL specifications. In L. Claesen, editor, *Applied Formal Methods for Correct VLSI Design*, Leuven, Belgium, November 1989. Elsevier Science Publishers.
- [16] P. J. Windley. *The Formal Verification of Generic Interpreters*. PhD thesis, University of California, Davis, Division of Computer Science, June 1990.
- [17] P. J. Windley. Using correctness results to verify behavioral properties of microprocessors. In *Proceedings of the IEEE Computer Assurance Conference*, June 1991.

## A New Variable Testability Measure

M. Jamoussi, B. Kaminska, D. Mukhedkar  
Department of Electrical and Computer Engineering  
Ecole Polytechnique de Montréal  
P.O.Box 6079, Station A  
Montréal, Canada (H3C 3A7)

**Abstract-** In this paper, we propose a new Variable Testability Measure (VTM) for implementing testability at the high-level synthesis stage of the design process of integrated circuits. This new approach, based on binary decision diagrams, representing fully functional blocks of a circuit, and on their cyclo-matic testability measures. It manipulates dataflow blocks to predict whether the circuit is testable and the vector set required to test it.

### 1 Introduction

In recent years, the use of silicon compilation and other standard cell design tools has changed the way digital systems are designed. As a result, more and more systems are being designed at the functional level, with little gate-level design being explicitly performed. So, an appropriate measure can be developed which efficiently represents knowledge about functional-level testability.

This paper addresses an approach for implementing testability in the high-level synthesis process, and particularly at the functional-level stage. Then, a new Variable Testability Measure (VTM) is defined and used to evaluate the dataflow testability in an advanced step of the design process.

### 2 Variable Testability Measure

We are interested in the testability of integrated circuits as early as possible in their design process.

Usually, testability implementation is left until after the design is completed, which requires greater effort at later stages. Testability analysis tools, such as SCOAP [7], which are supposed to support testability incorporation during the design stages, actually provide poor predictions of testability and do not suggest how and what test methodologies should be applied.

#### 2.1 High-Level Synthesis

Synthesis involves finding a structure that implements the behavior, the constraints and the goals of a given system. Generally, synthesis may be considered at various levels of abstraction because designs can be described at various levels of details. High-level synthesis [9] is the type of synthesis that begins at what is often called the algorithmic level.

It takes an abstract behavioral specification of a digital system and finds a register/transfer-level structure to realize the given behavior.

The system to be designed is usually represented at the algorithmic level by a programming language such as PASCAL [10] or ADA [6], or by a hardware description language similar to a programming one, such as VHDL [8]. The first step in high-level synthesis is usually the compilation of the formal language describing the system behavior into an internal representation.

The next two steps in synthesis transform core behavior into structure: scheduling and allocation. They are closely interrelated and interdependent. Scheduling consists in assigning the operations to so called control steps, fundamental sequencing units in synchronous systems and corresponds to a clock cycle. Allocation consists in assigning the operations to hardware. Finally, the design has to be converted into real hardware. Lower-level tools such as logic synthesis and layout synthesis complete the design.

The advantages of implementing testability as early as possible in the design process are a testable design, a reduced test cost and an earlier detection of intestable blocks. To incorporate testability in high-level synthesis as a constraint of the design specifications, a new concept of Variable Testability Measure is introduced providing information if the circuit is testable and permits a good prediction of the test-vectors set for a graph-representation of a circuit.

## 2.2 Variable Testability Measure

A new method for gathering the testability information at the dataflow-design stage called the Variable Testability Measure (VTM) is introduced. It permits an easy propagation of the information about functional-block testability and indicates that testability problems are very easily dealt with high-level synthesis.

Using a hierarchical abstraction principle, VTM will be able to provide two kinds of informations: first, whether or not each subcircuit and then the whole circuit are testable; second, the minimum number of test vectors required for a specific block of the circuit, and then for the whole circuit. This approach is based on the binary decision diagram [1] and the cyclomatic testability measure [2].

The Binary Decision Diagram (BDD) is a method for defining, analyzing, testing, and implementing large digital functions. It provides a complete implementation-free description of the functions involved. One of the areas in which these diagrams can be particularly useful is that of test generation, i.e. finding a set of inputs able to confirm that a given implementation performs correctly. Finally, BDDs may be directly interconnected to define still larger functions.

The Cyclomatic Testability Measure (CTM) is a method for predicting and determining a minimum set of test vectors for graphs of combinational and sequential circuits in the early conceptual stage of the design process. This approach is based on the cyclomatic number [3], and the BDD. If we note  $V$  the CTM of a given BDD called  $G$ ,  $V$  is computed by the following equation:

$$V(G) = e - n + 3 \quad (1)$$

where  $e$  and  $n$  are the numbers of edges and nodes in the graph  $G$  respectively.

Finally, it was shown that the CTM, called  $V$ , of a circuit composed of  $p$  subcircuits and represented by a BDD called  $G$ , is computed by the following equation:

$$V(G) = V(G_1) + \sum_{i=2}^p (V(G_i) - 2) \quad (2)$$

where  $V(G_1)$  is the CTM of the subcircuit containing the entry node represented by the BDD, called  $G_1$ .  $V(G_i)$  corresponds to the  $i^{th}$  subcircuit represented by its BDD, called  $G_i$ , ( $i = 2, \dots, p$ ).

## 2.3 Definition of VTM

The concept of VTM is a further development of the Cyclomatic Testability Measure. The VTM is defined as follows: Consider a functional block having its input and output variables given on  $n$  bits, such as an adder or a multiplier. VTM is a coefficient assigned to each bit of the input and output variables of this functional block. The VTM of a bit means the minimum number of test vectors required to test it. In this effect a variable given on  $n$  bits has  $n$  different VTMs, one for each bit.

The VTM permits treatment of functional blocks having their inputs and outputs given on various numbers of bits, while the CTM treats blocks with single outputs and inputs given on one bit each of them.

The advantage of the VTM is the easy composition of various blocks, which permit testability incorporation in the dataflow of the synthesis process.

## 3 Use of VTM in High-Level Synthesis

Throughout this paragraph, we try to use this new measure for some common functional primitives. Next, we will see how this new measure is involved in the evaluation of the testability of a circuit in its high-level synthesis stage.

### 3.1 Computation of VTM for some functional blocks

We try to determine the VTMs of the outputs of some functional-level circuit primitives such as comparators, adders, multipliers, logical operators, multiplexors which basically form the data flow of a circuit. Three cases are discussed in this section.

For the cases of functional primitives studied, we note  $A$  and  $B$  the inputs of these blocks, each one is given on  $n$  bits:  $A_n A_{n-1} \dots A_i \dots A_2 A_1$  and  $B_n B_{n-1} \dots B_i \dots B_2 B_1$  respectively. This notation means that  $A_n, A_{n-1}, \dots, A_i, \dots, A_2, A_1$  and  $B_n, B_{n-1}, \dots, B_i, \dots, B_2, B_1$  are the binary encoding of  $A$  and  $B$  respectively. We also note  $a_1, a_2, \dots, a_i, \dots, a_n$  and  $b_1, b_2, \dots, b_i, \dots, b_n$  the VTMs of  $A$  and  $B$  respectively.

### 3.2 A Comparator

Let us consider the case of a comparator of two variables  $A$  and  $B$ , given on  $n$  bits. The logical value of the output  $S$  is 1, while  $A$  is greater than  $B$  ( $A > B$ ). If not, the value of  $S$  is 0.

In figure 1, we present the BDD describing this functional block. According to equations (1) and (2), the VTM of  $S$ , noted  $s$ , is:

$$s = (4 * n + 3) - (a_n + b_1) + 2 * \sum_{i=1}^n (a_i + b_i + 4) \quad (3)$$

So the comparator of two variables, given on  $n$  bits each one, is tested with a minimum number of test vectors found by equation (3).

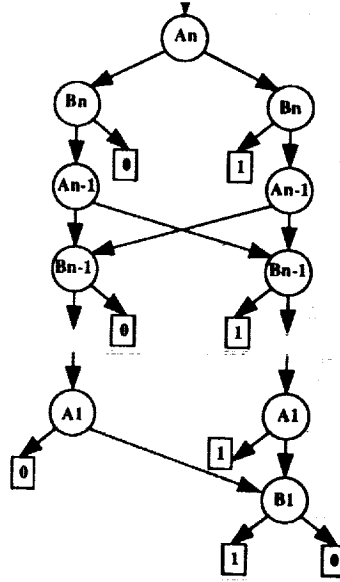


Figure 1: BDD of a Comparator

### 3.3 A Multiplexor

Let us study now the case of a multiplexor of two variables. The inputs  $A$ ,  $B$  are given on  $n$  bits while the control signal  $C$  on one bit. The output  $S$ , which is  $S_n S_{n-1} \dots S_i \dots S_2 S_1$ , is equal to  $A$  if  $C$  is true, or else it is equal to  $B$ .

The multiplexor can be considered in this case as a set of elementary multiplexors, where the  $i^{th}$  bit of the output is given as follows, ( $i = 1, \dots, n$ ):

$$S_i = C.A_i + \bar{C}.B_i \quad (4)$$

The CTM of the BDD describing  $S_i$  is equal to 4. Then, supposing  $a_i, b_i, c, s_i$  the VTMs of  $A_i, B_i, C, S_i$  respectively, according to equations (1) and (2),  $s_i$  is computed as follows, ( $i = 1, \dots, n$ ):

$$s_i = a_i + b_i + c + 2 \quad (5)$$

Finally, if  $A$  and  $B$  are primary inputs, which means  $a_i = b_i = 2$ , ( $i = 1, \dots, n$ ), according to (5),  $s_i$  is given as follows:

$$s_i = c + 2 \quad (6)$$

### 3.4 An (n,n) Adder

Let us consider now, an adder of two variables  $A$  and  $B$  given on  $n$  bits, called an (n,n) adder. The sum  $S$  is given on  $(n + 1)$  bits and is  $S_{n+1}S_n..S_i..S_2S_1$ . Considering the elementary full-adder, we have:

$$S_i = A_i \oplus B_i \oplus R_{i-1} \quad (7)$$

$$R_i = M(A_i, B_i, R_{i-1}) \quad (8)$$

where  $S_i$  is the  $i^{th}$  bit of the sum and  $R_i$  the carry out of this addition. The CTMs of the BDDs describing  $S_i$  and  $R_i$  are 7 and 5 respectively. Then, if  $s_i$ ,  $r_{i-1}$  and  $r_i$  are the VTMs of  $S_i$ ,  $R_{i-1}$  and  $R_i$  respectively, according to equation (2)  $s_i$  and  $r_i$  are computed as follows, ( $i = 2, \dots, n - 1, n$ ):

$$s_i = 7 + (a_i - 2) + (b_i - 2) + (r_{i-1} - 2) \quad (9)$$

$$r_i = 5 + (a_i - 2) + (b_i - 2) + (r_{i-1} - 2) \quad (10)$$

In the case of the half-adder, we have:

$$S_1 = A_1 \oplus B_1 \quad (11)$$

$$R_1 = M(A_1, B_1, 0) \quad (12)$$

Then, supposing  $s_1$  and  $r_1$  the VTMs of  $S_1$  and  $R_1$  respectively,  $s_1$  and  $r_1$  are computed as follows:

$$s_1 = 4 + (a_1 - 2) + (b_1 - 2) \quad (13)$$

$$r_1 = 3 + (a_1 - 2) + (b_1 - 2) \quad (14)$$

Noting  $a_p$  and  $b_p$  the VTMs of  $A_p$  and  $B_p$  respectively ( $p = 1, \dots, i$ ), the VTM  $s_i$  of  $S_i$ , the  $i^{th}$  bit of  $S$ , is:

$$s_i = \sum_{p=1}^i (a_p + b_p) + (2 - i) \quad (15)$$

OPERATOR	K (2 bits)	K (3 bits)
Addition	12	23
Subtraction	12	23
Comparator	7	11
Multiplexor	8	12
Multiplication	9	24
AND / OR	3	3

Table 1: Operator-Cost Coefficients

### 3.5 Objective function

It is common practice to use the cost function in the high-level synthesis, considering delay, area for testability constraints [5]. In this work, we propose a new objective function able to estimate and evaluate particularly the testability and the area constraints of a circuit. In a first stage, we propose to define a coefficient  $K$  for each functional primitive as the sum of its output VTMs, while it only has primary inputs. the value of  $K$  depends essentially on the bit number of the given functional-primitive variables.

In the case of logical operators or specific blocks where variables are given on one bit such as AND, OR gates,  $K$  is the output VTM. Table 1 gives the costs of some common functional blocks operating with variables given on 2 bits and 3 bits respectively. This coefficient will be, for a given functional primitive, its cost coefficient in the objective function introduced below:

### 3.6 Definition

Let us consider a circuit  $C$  composed of  $n$ -connected functional primitives. Given the  $i^{th}$  functional block ( $i = 1, \dots, n$ ), let us assume:

- $m_i$ : the sum of the bit numbers of its outputs.
- $a_{i,p}$ , ( $p = 1, \dots, m_i$ ): the VTMs of this block outputs.
- $K_i$ : the cost of this block.

The objective function is defined as follows:

$$f = \sum_{i=1}^n K_i * \left( \sum_{p=1}^{m_i} a_{i,p} \right) \quad (16)$$

The function given by equation (16) shows a trade-off between the circuit area (functional primitives used), and the number of test vectors or, in other words, the test time. One of our goals then, by using this new measure in high-level synthesis, can be expressed as a question of minimizing this objective function.



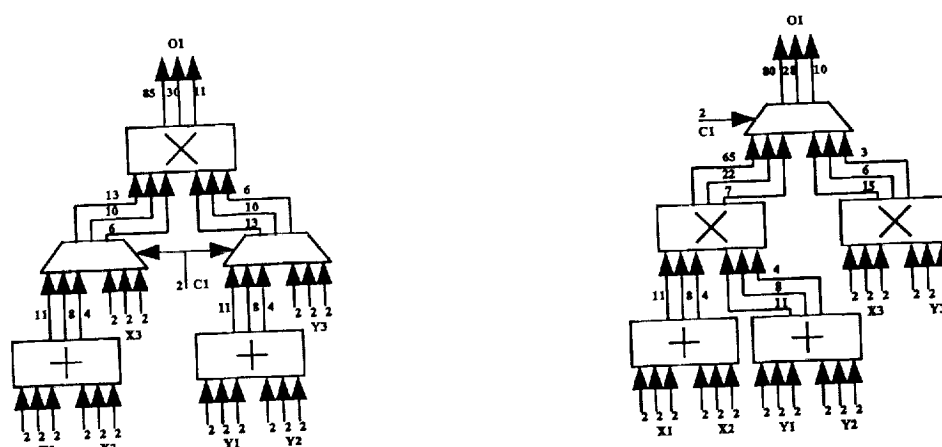


Figure 2: (a) Example circuit (b) Example circuit modified

The example presented in figure 2(a) is modified in 2(b) without changing the main task of the circuit. We notice lower VTM values against a greater silicon area due to a second multiplier used in figure 2(a). This transformation shows the trade-off discussed above.

## 4 Conclusion

In this paper we have proposed a new Variable Testability Measure (VTM) with the aim of incorporating testability at the functional-level stage, considered as an advanced step in the design process of a circuit. Our approach is essentially based on the Binary Decision Diagram (BDD) and the Cyclomatic Testability Measure (CTM). We have proposed an objective function to estimate circuit-testability cost.

## References

- [1] S. B. Akers, Binary Decision Diagram, *IEEE Transactions on Computers*, Vol. C-27, pp. 509-516, No.6, June 1978.
- [2] B. Ayari & B. Kaminska, A Cyclomatic Testability Measure, *Proceedings of the Canadian Conference on VLSI*, Ottawa, pp. 8.2.1-8.2.10, 1990.
- [3] C. Berge, Graphs and Hypergraphs, North-Holland, 1973.
- [4] M. A. Breuer & T. H. Chen, Automatic Design for Testability Via Testability Measures, *IEEE Transactions on Computer-Aided Design*, Vol. CAD.4, pp. 3-11, January 1985.

- [5] M. I. Elmasry & C. H. Gebotys, VLSI Design with Testability, *Design Automation Conference*, pp. 16-21, 1988.
- [6] E. F. Girczyc, Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Description, PhD Thesis, Carleton University, July 1984.
- [7] L. Goldstein & E. L. Thigpen, SCOAP: Sandia Controllability / Observability Analysis Program, *IEEE Design Automation Conference*, 1980.
- [8] A. Lowenstein & G. Winter, VHDL 's Impact on Test, *IEEE Design & Test Computers*, V3 n2, pp. 48-53, April 1986.
- [9] A. C. Parker, Tutorial on High-Level Synthesis, *IEEE Design Automation Conference*, pp. 330-336, 1988.
- [10] Trickey & H. Flamel, A High-Level Hardware Compiler, *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6,2 pp. 259-269, March 1987.

# Controlling State Explosion During Automatic Verification of Delay-Insensitive and Delay-Constrained VLSI Systems Using the POM Verifier <sup>1</sup>

D. Probst and L. Jensen  
Department of Computer Science  
Concordia University  
1455 de Maisonneuve Blvd. West  
Montreal, Quebec Canada H3G 1M8

*Abstract*— Delay-insensitive VLSI systems have a certain appeal on the ground due to difficulties with clocks; they are even more attractive in space. We answer the question, is it possible to control state explosion arising from various sources during automatic verification (model checking) of delay-insensitive systems? State explosion due to concurrency is handled by introducing a partial-order representation for systems, and defining system correctness as a simple relation between two partial orders on the same set of system events (a graph problem). State explosion due to nondeterminism (chiefly arbitration) is handled when the system to be verified has a clean, finite recurrence structure. Backwards branching is a further optimization. The heart of this approach is the ability, during model checking, to discover a compact finite presentation of the verified system without prior composition of system components. The fully-implemented POM verification system has polynomial space and time performance on traditional asynchronous-circuit benchmarks that are exponential in space and time for other verification systems. We also sketch the generalization of this approach to handle delay-constrained VLSI systems.

Keywords: delay-insensitive system, model checking, state explosion, partial-order representation, recurrence structure, state encoding, delay-constrained reactive system.

## 1 Introduction

Delay-insensitive systems are motivated by difficulties with clock distribution and component composition in clocked systems [1,2,5,9]. In a delay-insensitive system, modules may be interconnected to form systems in such a way that system correctness does not depend on delays in either modules or interconnection media. Gate-level implementations of modules whose specifications are delay-insensitive are often themselves quasi-delay-insensitive; essentially, the assumption of isochronic forks allows one gate to handshake on behalf of

---

<sup>1</sup>This research was supported by the Natural Sciences and Engineering Research Council of Canada under grants A3363 and MEF0040121. Email: probst@crim.ca.

another. Most interesting are delay-constrained reactive systems, in which either outputs or inputs or both must appear in some temporal window relative to enabling inputs or outputs. Hardware systems in space make delay insensitivity even more attractive due to (i) pervasive asynchronous communication, and (ii) extremely-low-power applications. Delay insensitivity has a natural link to controlling state explosion during automatic verification; the simple enabling relations in delay-insensitive control systems make it easy to discover a solution to the state-explosion problem based on causality checking. To build an automatic verifier based on causality checking, you need two things: (i) an expressive finite partial-order representation strategy that explicitly distinguishes concurrency, choice and recurrence, and (ii) a “goal-directed” state-encoding strategy that is both comprehensive (includes all causality) and minimal (has fewest states)—the last for performance reasons. Given these two things, you can combine the best features of automata-based and partial-order-based computational verification methods.

## 2 Behavior Automata

The basic automata used to represent processes are called behavior automata, which can be unrolled to produce event structures (essentially sets of partially-ordered computations with all branching due to conflict resolution made explicit) [5-8]. Partial orders and concurrent computation are discussed in [3]. Restrictions on behavior automata trade off between expressiveness and processability (e.g., the efficiency of verification algorithms) [8]. The most important rules for delay insensitivity are (cf. [10]):

**Rule 1** Any two events at the same port in a partially-ordered computation are order-separated by at least one event at some other port.

**Rule 2** There is no immediate order relation between two input events or two output events. Each ordering chain is an infinite sequence of strictly alternating input and output events.

We seek abstract, i.e., black-box, specifications [4]. For this purpose, behavior automata are constructed in three phases. First, there is a deterministic finite-state machine (stick figure) that expresses both conflict resolution (choice) and recurrence structure. This is a “small” automaton relative to the full transition system. Second, there is an expansion of dfsm transitions (sticks) into finite posets, with additional machinery (sockets) to define possibly nonsequential concatenation of posets. Third, there is an iterative process of labeling successor arrows in posets, which terminates with an appropriate state encoding.

We sketch the formal definition of behavior automaton. Given disjoint alphabets  $Act$  (process actions),  $Arr$  (successor-arrow labels),  $Com$  (dfsm transitions) and  $Soc$  (sockets), first define  $Pos$  as the set of finite labeled posets over  $Act \cup Soc$ . Each member of  $Pos$  is a labeled poset  $(B, \Gamma, \nu)$ , where (i)  $\Gamma$  is a partial order over  $B \subseteq Act \cup Soc$ , and (ii)  $\nu: \Omega \rightarrow Arr$  assigns a label to each element in the successor relation  $\Omega$  (the transitive reduction of  $\Gamma$ ). A behavior automaton is a 3-tuple  $(D, \eta, o)$ , where (i)  $D$  is a dfsm over  $Com$ , (ii)

$\eta$ :  $\text{Com} \rightarrow \text{Pos}$  maps dfsm transitions to labeled posets, and (iii)  $\circ$ :  $\text{Soc} \rightarrow \text{powerset}(\text{Act})$  maps sockets to sets of process actions. Map  $\circ$  defines which process actions can “plug in” to an empty socket when a poset command is concatenated to a sequence of earlier poset commands as defined by dfsm  $D$ .

A C-element has two input ports  $a$  and  $b$ , and an output port  $c$ . Two actions are possible at a given port depending on whether the signal transition is rising (+) or falling (−). There is no conflict resolution (choice), and the recurrence structure of  $D$  is a simple loop. Transitions (sticks) concatenate sequentially in this example, shown in Fig. 1. Both the reset action and action  $c^-$  can fill the unique socket in this poset. Digit colons identify dfsm  $D$  vertices.

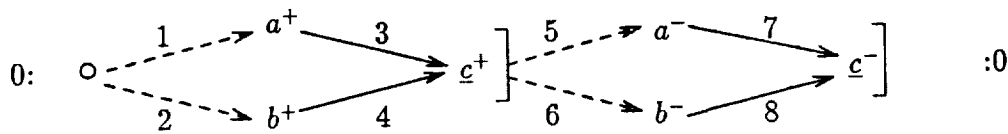


Figure 1: Behavior automaton for a C-element.

In the absence of conflict resolution, each enabled output action must be performed eventually (indicated by bracketing). The use of both dashed and solid arrows is a visual reminder that a process specification contains both an interprocess protocol (given by the dashed arrows) and an intraprocess protocol (given by the solid arrows). Here, the state encoding (arrow labeling) is essentially fixed; since the state is encoded as the set of successor arrows crossing from the past to the future, i.e., crossing a consistent cut produced by a partial execution, using fewer arrow labels would alter the enabling relation of the C-element.

The semantics are straightforward. For example, action  $a^+$  is enabled in any state containing arrow 1; when it is performed, arrow 1 is removed from the state and arrow 3 is added. Similarly, action  $c^+$  is enabled and required (because of the bracket) in any state containing arrows 3 and 4. When it is performed, arrows 3 and 4 are removed from the state and arrows 5 and 6 are added. Action  $c^-$  has preset and postset given by:  $\{7, 8\} c^- \{1, 2\}$ .

Behavior automata are more interesting when branching is involved. A delay-insensitive arbiter has two input ports  $a$  and  $b$ , and two output ports  $c$  and  $d$ . It grants exclusive access to one of two competing clients at a time. The behavior automaton is shown in Fig. 2.

Clients follow a four-cycle protocol.  $(A) = c^+ \rightarrow a^-$  and  $(B) = d^+ \rightarrow b^-$  are the two critical sections. The labeling shown, if completed, would be conservative (the state encoding includes all causality, but is not minimal). Having arrows 8, 9 and 10 in state encodings indicates who made the token available (viz., first client, second client and reset action). These three arrows are distinct instances of causality that must be checked separately. Still, there are too many state encodings.

We can group arrows 8, 9 and 10 into an equivalence class  $t$ . This does not alter the enabling relation. Consider performing action  $c^+$  in state  $\{1, 5, t\}$ . Causality checking

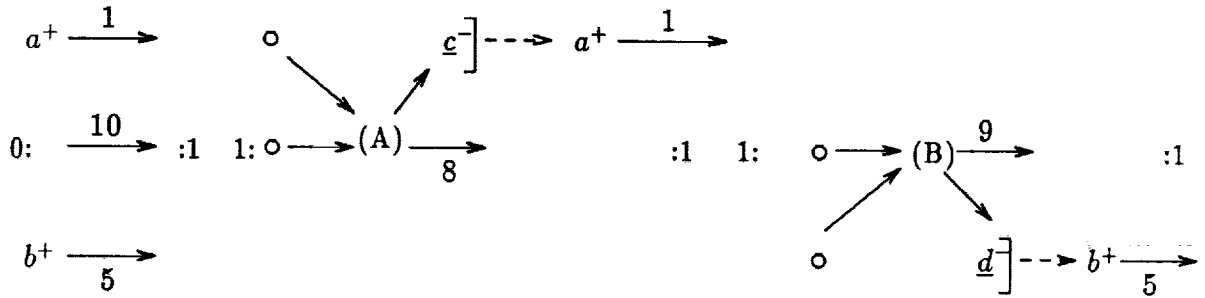


Figure 2: Behavior automaton for a delay-insensitive arbiter.

of arrow  $t$  requires backing up in the behavior automaton to both possible sources, viz., actions  $a^-$  and  $b^-$ . In state  $\{1, 5, t\}$ ,  $c^+$  and  $d^+$  are concurrently enabled but conflicting actions. Verification algorithms that process behavior automata perform both forwards branching (conflict resolution) and backwards branching (examination of distinct recent pasts).

After equivalenced arrow  $t$  has been defined, we can complete the picture in Fig. 2 to make it match the formal definition (the labeled arrows leaving posets are derivable from map  $o$ ). Consider the second poset command. The top socket is filled only by action  $a^+$ ; its arrow is labeled 1. The middle socket is filled by any of the actions  $a^-$ ,  $b^-$  and reset; its arrow is labeled  $t$ . The remaining (interior) poset arrows are given arbitrary distinct labels.

### 3 Correctness as a Graph Problem

We define correctness by using the mirror  $mP$  of specification  $P$  as a conceptual implementation tester [1]. We form an imaginary closed system  $S$  by linking mirror  $mP$  of specification  $P$  to the implementation network of processes  $Net$ . This produces an infinite pomtree (event structure) of system events on which two partial orders are defined; system correctness is then expressible as a simple, easily-checked relation between the partial orders. The standard model-independent notion of correctness is as follows. Is there a failure somewhere, causing system  $S$  to become undefined? Does the system just stop, violating fundamental liveness? Is some progress requirement of  $P$  violated? Is there (program-detectable) nondeterminate livelock in  $S$  so that an appeal to fairness of system components is necessary to assert progress? Is some conflict corresponding to output choice in  $P$  resolved unfairly?

Mirror  $mP$  is formed by inverting the type of  $P$ 's actions and the causal/noncausal interpretation of  $P$ 's successor arrows, turning  $P$ 's dashed arrows into solid arrows and vice versa. Brackets are preserved unchanged. Every action that can be performed in  $S$  is a linked (output action, input action) pair. As a result, we can check whether intraprocess protocols support interprocess protocols in closed system  $S$ .

We bootstrap the dashed (noncausal, interprocess protocol) and solid (causal, intraprocess protocol) relations from process actions to system actions, defining an event structure

(sometimes called pomtree) with a noncausal enabling relation on top of the usual causal enabling one. For example, a noncausal predecessor of system action  $\sigma$  is found by locating the embedded process input action, stepping back along a dashed process arrow, and returning to the system alphabet. We have thus defined "noncausal preset" of a system action. Essentially, the safety correctness relation is: whenever a dashed arrow links two system actions, a chain of solid arrows must also link the two actions.

Let  $\sigma$  be a system action that is causally enabled in  $S$ . There is a safety violation at  $\sigma$  unless

- (a) its noncausal preset is also causally enabled in  $S$ , and
- (b) each member of its noncausal preset is a causal ancestor of  $\sigma$ .

The causal preset of  $\sigma$  is defined only when  $\sigma$  is a bracketed system action: it is the set of nearest performances of linked mP output actions on any causal chain coming into  $\sigma$ . In order that a bracketed  $\sigma$  in  $S$  is neither a safety nor a progress violation, it is necessary that the causal and noncausal presets of  $\sigma$  match exactly. When backwards branching is present in  $S$ , these conditions are generalized to hold along each distinct past (backwards branch). Backwards branching is necessary to resolve multiple sources of equivalenced arrows.

## 4 Model Checking

The algorithm is straightforward. Starting from system reset, we enumerate causally-enabled system actions and visit one system cut per action. We consider each enabled action in a state produced by some partially-ordered past that we have generated. First, we repeatedly step back across single dashed arrows to compute the action's noncausal preset. Second, we repeatedly (finitely) chain back across multiple solid arrows to compute the action's partial causal ancestor set (or causal preset if the action is bracketed). When equivalenced arrows are encountered, we branch backwards to check each possible source. The speedup is due to two effects:

1. we effectively check cuts in the generated past that we have passed by without visiting, and
2. for equivalenced arrows, we effectively check cuts in pasts that we have not generated.

This kills state explosion due to concurrency and/or nondeterminism. We traverse each determinate segment (stick) of the implicitly constructed system behavior automaton (stick figure) precisely once. Backwards branching catches all causality that would have been visible had we traversed the system stick figure in some other way. Example system stick figures are shown in Fig. 3.

We keep the termination table small by making the mapping from  $P$  states to  $S$  states one-to-few rather than one-to-many. This is possible when all behavior automata have

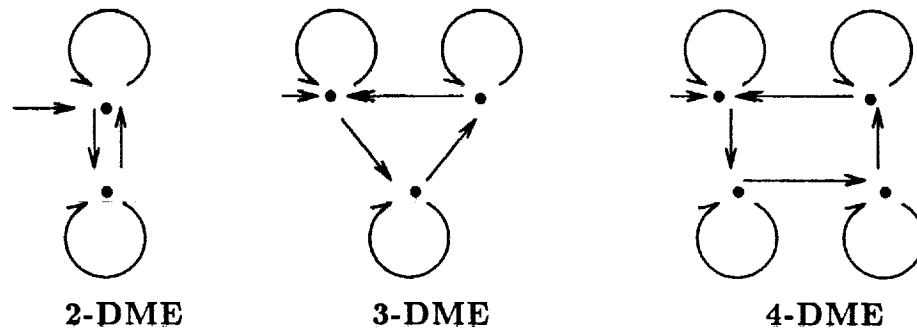


Figure 3: System stick figures for the  $n$ -DME verification problem.

visible branching and recurrence structure. Explicit structure in each component allows the verification algorithm to uncover a structure in system  $S$ . In particular, when we cycle in  $P$ , we can arrange to cycle in  $S$ . As a result, termination is achieved by checkpointing very few global states of system  $S$ . The top level of the algorithm visits system actions and tries to complete  $P$  sticks. The lower level of the algorithm does arrow checking.

## 5 Output-delay-constrained reactive systems

To fix ideas, consider a hardware system that is a space-based component of a missile defense system; this component receives massive amounts of target-acquisition data asynchronously, and is required to process it in real time and communicate the result. There are two types of delay constraint that could appear in a requirements specification of such a component, which is a typical reactive system. First, there could be a temporal interval, relative to the arrival of a complete problem instance, during which the component must respond; this is an output delay constraint. Second, there could be a temporal interval, relative to the departure of the previous result and/or the arrival of other input, during which the external world can safely stimulate the component; this is an input delay constraint. The simplest delay-constrained reactive systems are those in which delay constraints are imposed only on the intraprocess protocol, i.e., on module response; in this case, the mechanism that ensures input safety is unchanged (the interprocess protocol is still real or virtual handshaking). The difficult case is an interprocess protocol that specifies when the module can be overwhelmed by high-bandwidth input; we leave the difficult case for future work. In our representation, minimum/maximum-delay information is expressed by putting timing windows directly on output actions. Minimum-delay information may be freely entered on successor arrows, but maximum-delay semantics is constrained by questions of physical realizability. We choose the following uniform semantics. If bracketed output action  $c$  is annotated with the temporal interval  $(t_{\min}, t_{\max})$ , then action  $c$  will be performed no earlier than  $t_{\min}$  units and no later than  $t_{\max}$  units after the holding of its preset  $\text{pre}(c)$ .

The standard verification algorithm for precedence constraints (described in section 4) can easily be extended to check these new delay constraints. When checking for a



(precedence) safety violation at system action  $\sigma$ , we determine whether there is a causal chain to  $\sigma$  from each member of  $\sigma$ 's noncausal preset, say,  $\text{pre}(\sigma)$ . First, copy the timing window on each output action to each of its predecessor arrows. Second, find the sums of  $t_{\min}$  and  $t_{\max}$  along all causal chains to  $\sigma$  from each member of its noncausal preset  $\text{pre}(\sigma)$ . Consider the maximum delay case. For  $\tau \in \text{pre}(\sigma)$ , define  $D(\tau, \sigma)$  as the maximum sum of  $t_{\max}$  values along any causal chain from  $\tau$  to  $\sigma$ . Then system action  $\sigma$  will be performed no later than  $\max$  over  $\tau$  of  $D(\tau, \sigma)$  units after the holding of its noncausal preset  $\text{pre}(\sigma)$ . For the minimum delay case, define  $d(\tau, \sigma)$  as the maximum sum of  $t_{\min}$  values, and take the  $\min$  over  $\tau$  of  $d(\tau, \sigma)$ ;  $\sigma$  will be performed no earlier than this many units after the holding of its noncausal preset.

## 6 Conclusion

A complete verification package has been written by Lin Jensen in the Trilogy programming language running on an IBM PC. The POM system has polynomial space and time performance on benchmarks that are exponential in space and time for other verification systems. Consider the ring of DME elements benchmark. The runtime for verification of both safety and progress properties is quadratic in  $n$ , the number of DME elements. The number of system states grows exponentially with  $n$ . For example, when  $n = 9$ , the time is 180 s (roughly  $10^9$  states); when  $n = 10$ , the time is 220 s (roughly  $10^{10}$  states). The space requirements for these problems do not exceed 64K bytes, i.e., one IBM PC data segment. What are the compiler-independent space requirements? One must store the input; this is linear. One must store the termination table; this is quadratic. Given reasonable garbage collection, the working storage to do backwards chaining in a partially-ordered system computation is linear, because one constructs and compares simple presets. The limiting resource is the quadratic space used to store the termination table. To repeat, both space and time are quadratic, in this example, to verify a concurrent system with exponentially many states. Building up the actual partially-ordered system computations themselves is unnecessary; we work directly with the uncomposed behavior automata of the system components. We have also shown, at least in the simple case of output-delay-constrained reactive systems, that verifying temporal window constraints is barely more expensive than verifying precedence constraints. In general, the achievable efficiency of a real-time verification algorithm is a sensitive function of the precise abstraction of real time used in the model.

## References

- [1] D.L. Dill, "Trace theory for automatic hierarchical verification of speed-independent circuits", PhD Thesis, Department of Computer Science, Carnegie Mellon University, Report CMU-CS-88-119, February 1988. Also MIT Press, 1989.

- [2] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits", *Distributed Computing*, Vol.1, No.4, October 1986, pp. 226-234.
- [3] V. R. Pratt, "Modeling concurrency with partial orders", *Int. J. of Parallel Prog.*, Vol.15, No.1, February 1986, pp. 33-71.
- [4] D. K. Probst and H. F. Li, "Abstract specification of synchronous data types for VLSI and proving the correctness of systolic network implementations", *IEEE Trans. on Computers*, Vol. C-37, No. 6, June 1988, pp. 710-720.
- [5] D. K. Probst and H. F. Li, "Abstract specification, composition and proof of correctness of delay-insensitive circuits and systems", Technical Report, Department of Computer Science, Concordia University, CS-VLSI-88-2, April 1988 (Revised March 1989).
- [6] D. K. Probst and H. F. Li, "Partial-order model checking of delay-insensitive systems". In R. Hobson et al. (Eds.), *Canadian Conference on VLSI 1989, Proceedings*, Vancouver, BC, October 1989, pp. 73-80.
- [7] D. K. Probst and H. F. Li, "Using partial-order semantics to avoid the state explosion problem in asynchronous systems". In E. M. Clarke and R. P. Kurshan, (Eds.), *Workshop on Computer-Aided Verification '90, DIMACS Series*, Vol. 3, 1991, pp. 15-24. Also *Lect. Notes in Comput. Sci.*, Springer Verlag, forthcoming.
- [8] D. K. Probst and H. F. Li, "Partial-order model checking: A guide for the perplexed". In K. G. Larsen and A. Skou, (Eds.), *Workshop on Computer-Aided Verification '91, Proceedings*, Department of Mathematics and Computer Science, Aalborg University, Report IR-91-5, July 1991, pp. 405-416. Also *Lect. Notes in Comput. Sci.*, Springer Verlag, forthcoming.
- [9] J.v.d. Snepscheut, "Trace theory and VLSI design", *Lect. Notes in Comput. Sci.* 200, Springer Verlag, 1985.
- [10] J.T. Udding, "A formal model for defining and classifying delay-insensitive circuits", *Distributed Computing*, Vol. 1, No. 4, October 1986, pp. 197-204.

# Formal Verification of an MMU and MMU Cache

E. T. Schubert

Division of Computer Science

University of California, Davis

**Abstract** - We describe the formal verification of a hardware subsystem consisting of a memory management unit and a cache. These devices are verified independently and then shown to interact correctly when composed. The MMU authorizes memory requests and translate virtual addresses to real addresses. The cache improves performance by maintaining a LRU list from the memory resident segment table.

## 1 Introduction

Computers are being used in areas where no affordable level of testing is adequate. Safety and life critical systems must find a replacement for exhaustive testing to guarantee their correctness. Through a mathematical proof, hardware verification can formally demonstrate that a design satisfies its specification. However, hardware verification research has focused on device verification and has largely ignored system composition verification [1]. Our research is directed towards developing a methodology to verify a hardware base for a safety critical system. The top level hardware specification is apt to suggest a unitary implementation. This abstraction is convenient for verifying the correctness of software, however, the implementation consists of many different interacting components (CPU, memory, coprocessors, I/O devices, bus controllers, interrupt controllers, etc). This paper will describe our efforts to verify a subsystem consisting of a MMU and its cache using the HOL theorem prover [2].

The abstract MMU reported in [3] assumed a memory model where a read request was satisfied in one cycle. We extend the MMU to interact with an asynchronous memory. Additionally, the memory is more fully described; providing read and write functions. These changes required several significant changes to the abstract MMU proof script. The original proof strategy took advantage of the single cycle response time. The new strategy must use two arbitrary contents to define when memory words are returned from the memory-cache subsystem.

### 1.1 Related Work

Hardware verification requires that the design of a system is formally shown to satisfy its specification through a mathematical proof. Using theorem proving techniques, an expression describing the behavior of a device is proven to be equivalent in some sense to an expression describing the implementation structure of the device. These expressions concisely describe the behavior of devices in an unambiguous way. An additional benefit of hardware verification is that the behavioral semantics of the hardware are clearly defined. This provides an accurate basis for building correct software systems [5].

Hardware verification efforts thus far have focused primarily on a microprocessor as the base for computer systems [6], [7], [8], [9]. The processors verified have modeled small instruction sets and generally, have not included modern CPU features such as pipelines, multiple functional units and hardware interrupt support. Tamarack-3 [9] and AVM-1 [10] do provide sufficient interrupt support to connect with an interrupt controller. However, no system currently verified provides the memory management functions necessary to support a secure operating system.

Previous efforts to verify systems have constructed *vertically verified systems* with a microprocessor/memory as the system's base [11],[5],[1]. These efforts have aimed at illustrating how hardware verification can be used to close the semantic gap between high level languages and the computer's instruction set. However, the base for these systems (a microprocessor-memory pair) has been an unrealistic hardware platform.

## 1.2 HOL

The object language of HOL is a formulation of higher-order logic. Universally quantified variables are used to specify input and output device lines while internal device lines are existentially quantified. Conditional expressions are in the form: `cond → then-clause | else-clause`.

HOL provides the human verifier with a selection of tactics for use in goal-directed proofs. The tactics are very similar to the kinds of steps a human theorem prover would take in solving a goal. New tactics can be written that allow the theorem prover to be extended and customized for a particular task. New theorems can only be created in a controlled manner. All proofs can be reduced to one containing only the 8 primitive inference rules and 5 primitive axioms. High-level inference rules and tactics derived from some combination of primitive inference rules.

The following HOL expression defines an and gate implementation using an inverter and a nand gate. The existentially quantified variable *p*, represents an internal line which links the output of the nand gate with the input of the inverter.

$$\vdash_{def} \text{andGate } a \ b \ \text{out} = \exists p. \text{nand } a \ b \ p \ \wedge \ \text{inv } p \ \text{out}$$

## 2 Memory Management Unit

[12] describes a number of memory management units which form a complexity hierarchy. By developing a sophisticated MMU in steps, the construction of the final proof appears to be more tractable. The simpler devices validate access to fixed length memory pages while the more complex devices authorize read, write or execute access to variable length segments and translate virtual addresses to real addresses. Many of these devices were designed and verified to the gate level. However, as the complexity increases, the emphasis of the verification shifts from gate level connections to the correctness of the operating system support features.

The device described below validates memory requests based on information maintained in a memory resident segment descriptor table. The location of the table is determined by a

segment table pointer register which is accessible only during supervisor operations. Each descriptor consists of two words: the first contains access control information (present bit, read/write/execute permissions, segment size) and the second serves as the base address for the segment's real location in memory. To translate from a virtual address to a real address, the MMU adds the segment offset to the segment base address. The MMU assumes the table provides an entry for all possible segment descriptors.

A generic theory for a class of MMU devices is defined where several functions and data types are left abstract. Using an abstract representation, details such as word length, can be omitted and the verification focuses only on the correctness of higher level abstraction (e.g. electronic block level rather than gate level). At a later point, the abstract representation can be instantiated with components that implement concrete behavior.

Support for generic or abstract theories is not directly provided by HOL. However, a theory about abstract representations can be defined in the object language [10]. An *abstract representation* contains a set of uninterpreted constants, types, abstract operations and a set of abstract objects. The semantics of the abstract representation is unspecified. Inside the theory, we do not know what the objects and operations mean. The abstract theory package also creates a set of selector functions [11] to extract desired functions from an abstract representation.

The abstract MMU representation generalizes traits particular to concrete implementations. Properties such as the the exact security policy and division of a virtual address into a segment identifier and offset (as well as the overall number of bits in an address), are hidden by functions which given an address, return the segment identifier or segment offset field (`segId` and `segOfs`, respectively). There is also a function `segIdshf` which returns the offset of a segment descriptor within the memory resident segment table for a given address. Since descriptors require two words, the implementation of this function simply shifts the segment identifier to the left one bit position (e.g. it adds a trailing zero bit).

The abstract functions selected by `availBit`, `readBit`, `writeBit` and `execBit` extract a bit value from an argument of type `*wordn`. These functions are applied to the first word of a segment descriptor.

Several functions which operate on two-tuples are available. Given a pair of `*wordn` values, `add` returns a value of `*wordn`. Functions `addrEq`, `ofsLEq` and `validAccess` replace the `bitVector` comparison units defined for the more concrete units.

Additional abstract coercion functions are available to convert values between types. If the theory were instantiated, the abstract types would likely be implemented with `bitVec`s; leaving these functions unnecessary.

Memory is also treated abstractly. The abstract representation provides a fetch function `fetch`.

---

```

let mmu_abs = new_abstract_representation
[
  ('segId',      ":(*address -> *wordn)"      );
  ('segOfs',     ":(*address -> *wordn)"      );
  ('segIdshf',   ":(*address -> *wordn)"      );
  ('availBit',   ":(*wordn -> bool)"          );
  ('readBit',    ":(*wordn -> bool)"          );
  ('writeBit',   ":(*wordn -> bool)"          );
  ('execBit',    ":(*wordn -> bool)"          );
  ('add',        ":(*wordn # *wordn -> *wordn)" );
  ('addrEq',     ":(*address # *address -> bool)" );
  ('ofsLEq',     ":(*address # *wordn -> bool)" );
  ('validAccess', ":(*address # *wordn # RWE -> bool)";
  ('val',        ":(*wordn -> num)"          );
  ('wordn',      ":(num -> *wordn)"          );
  ('address',    ":(*wordn -> *address)"      );
  ('fetch',     ":(*memory # *address) -> *wordn");
];;

```

---

A type abbreviation *RWE* is also defined to be a three tuple of bit values. Selector functions *rBIT*, *wBIT* and *eBIT* access the first, second and third bits, respectively.

## 2.1 Specification

The specification is decomposed into several rules and ignores timing characteristics. The state and output environment of the MMU specification is a three-tuple consisting of a boolean acknowledgment, a memory address and the table pointer register value. The variable *r* in the definitions below is the abstract representation.

Functions *superMode* and *userMode* describe the behavior of the MMU when operating in their respective modes. *legalAccess* uses many of the abstract functions to fetch from memory the appropriate segment descriptor and compare it with the request's access parameters. *vToR* constructs a real address from a virtual address.

---


$$\vdash_{def} \text{legalAccess } r \text{ vAddr tblPtr rwe mem} = \text{let } a = (\text{fetch } r)(\text{mem}, (\text{address } r)((\text{add } r) (\text{segIdshf } r \text{ vAddr}, \text{tblPtr}))) \text{ in } ((\text{validAccess } r) (\text{vAddr}, a, \text{rwe}) \wedge (\text{ofsLEq } r) (\text{vAddr}, a))$$

$$\vdash_{def} \text{vToR } r \text{ vAddr tblPtr mem} = \text{let } a = (\text{fetch } r) (\text{mem}, (\text{address } r)((\text{add } r)((\text{wordn } r \text{ 1}), (\text{add } r)(\text{segIdshf } r \text{ vAddr}, \text{tblPtr})))) \text{ in } (\text{address } r) ((\text{add } r) (\text{segOfs } r \text{ vAddr}, a))$$

$$\vdash_{def} \text{superMode } r \text{ vAddr rwe tblPtrADDR tblPtr data mem} = ((\text{wBIT } r \text{ rwe}) \wedge (\text{addrEq } r (\text{vAddr}, \text{tblPtrADDR}))) \rightarrow (T, \text{vAddr}, \text{data}) - (T, \text{vAddr}, \text{tblPtr})$$

$$\vdash_{def} \text{userMode } r \text{ vAddr rwe tblPtrADDR tblPtr data mem} = \text{legalAccess } r \text{ vAddr tblPtr rwe mem} \rightarrow (T, (\text{vToR } r \text{ vAddr tblPtr mem}), \text{tblPtr}) - (F, \text{vAddr}, \text{tblPtr})$$

$$\vdash_{def} \text{mmu.spec } r \text{ vAddr rwe tblPtrADDR tblPtr data mem superv} = \text{superv} \rightarrow \text{superMode } r \text{ vAddr rwe tblPtrADDR tblPtr data mem} - \text{userMode } r \text{ vAddr rwe tblPtrADDR tblPtr data mem}$$


---

## 2.2 Implementation

The implementation is constructed from electronic block model components. These are defined as specifications for the behavior of a gate level implementation. Many of the devices specify their timing behavior as well. The building blocks consist of a security comparison unit, an address match unit, a memory fetch unit, an adder, registers, latches, muxes, and a control unit. Most of the device definitions are straight forward with the exception of the memory and the control unit. These two units will be described in greater detail.

---

```

 $\vdash_{def} \text{secUnit\_spec } r \ a \ b \ rwe \ ok = \forall t. \ ok \ (t+1) =$ 
 $((\text{validAccess } r) ((a \ t), (b \ t), (rwe \ t)) \wedge (\text{ofsLEq } r) ((a \ t), (b \ t)))$ 
 $\vdash_{def} \text{addUnit\_spec } r \ a \ b \ c = \forall t:\text{num}. \ c \ (t+1) = (\text{add } r \ (a \ t), (b \ t))$ 
 $\vdash_{def} \text{muxUnit\_spec } r \ a \ b \ out \ w = \forall t. (out(t+1)) = (w(t+1)) \rightarrow \text{address } r(b(t+1)) | (a \ t)$ 
 $\vdash_{def} \text{mux3Unit\_spec } a \ b \ c \ out \ w = \forall t:\text{num}.$ 
 $(out \ t) = (w \ t = 0) \rightarrow a \ t \mid (w \ t = 1) \rightarrow b \ t \mid c \ t$ 
 $\vdash_{def} \text{splitUnit\_spec } r \ virt \ id \ ofs = \forall t:\text{num}.$ 
 $((id \ t) = (\text{segIdshf } r) (virt \ t)) \wedge ((ofs \ t) = (\text{segOfs } r) (virt \ t))$ 
 $\vdash_{def} \text{latchUnit\_spec } r \ i \ out \ ctrl = \forall t:\text{num}.$ 
 $out \ (t+1) = ctrl \ (t+1) \rightarrow out \ t \mid (i \ (t+1))$ 
 $\vdash_{def} \text{regUnit\_spec } r \ i \ ld \ clr \ out =$ 
 $(\forall t:\text{num}. out(t+1) = (clr \ t \rightarrow (\text{wordn } r \ 0) \mid ld \ t \rightarrow i \ t \mid out \ t))$ 
 $\wedge (out \ 0 = (\text{wordn } r \ 0))$ 
 $\vdash_{def} \text{matchUnit\_spec } r \ a \ b \ m = \forall (t:\text{num}). \ m(t+1) = (\text{addrEq } r \ (a \ t, b \ t)) \rightarrow T \mid F"$ 

```

---

### Memory Unit

As a first step towards composing devices, the memory specification used for the MMU verification is significantly expanded from the model used in [3]. The earlier model assumed a read-only memory that returned a value one clock cycle after a request was made. The new model defines asynchronous read and write operations. This model makes an implicit assumption that each memory request is satisfied before the next request is generated. Most of the new proof effort centered on establishing the correctness of the MMU control unit with the new memory specification.

---

```

 $\vdash_{def} \text{memoryUnit\_spec } r \ req \ rwe \ addr \ data \ done \ mem =$ 
 $(done \ 0 = F) \wedge$ 
 $(\forall t. (req \ t) \rightarrow$ 
 $(\exists t'. \text{Next done } (t, t+t') \wedge$ 
 $(wBIT \ (rwe \ t)) \Rightarrow$ 
 $( (mem \ (t+t') = \text{store } r \ (mem \ t, addr \ t, data \ t) ) ) \mid$ 
 $( (data \ (t+t') = \text{fetch } r \ (mem \ t, addr \ t) ) \wedge$ 
 $(mem \ (t+t') = mem \ t) ) ) )$ 
 $\mid$ 
 $( (done \ (t+1) = F) \wedge$ 
 $(mem \ (t+1) = mem \ t) ) )$ 

```

---

### Control Unit

To process each memory request, the control unit will pass through several clocked phases. At each clock tick the control unit may change its phase depending on the results computed by the other internal units and the MMU input from the system bus. The control unit state is maintained by the variable *phase*. There are six distinct phases, however,

### 8.3.6

not all phases are executed for each request. Which phases are executed depends on the validity of the memory request. Request evaluation begins with the control unit in phase 0 and completes when phase 0 is again reached. A valid request will require five phases with a delay of at least one time unit before each phase change.

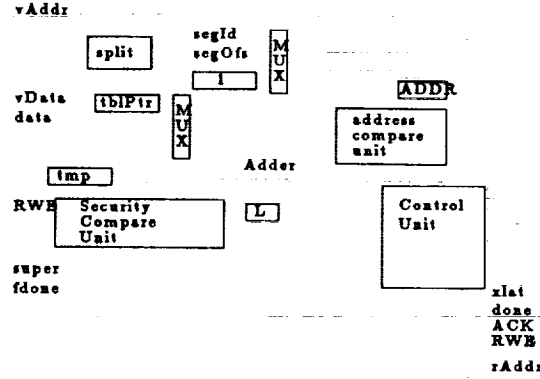


Figure 1: Abstract MMU Internal Block Diagram

The dataPath definition describes the interconnection between all the units other than the control unit.

```

def dataPath r vAddr vData rwe mem tblPtrADDR tblPtr rAddr muxC
    tmpC tblC lC rReq xlat match secOK fdone =
  ∃ (mux1 mux2 id ofs addOut data latOut secData.
    (regUnit_spec r vData tblC bitFalse tblPtr) ^
    (regUnit_spec r data tmpC bitFalse secData) ^
    (secUnit_spec r vAddr secData rwe secOK) ^
    (splitUnit_spec r vAddr id ofs) ^
    (mux3Unit_spec id ofs (oneUnit_spec r) mux1 muxC) ^
    (mux3Unit_spec tblPtr data latOut mux2 muxC) ^
    (addUnit_spec r mux1 mux2 addOut) ^
    (latchUnit_spec r addOut latOut lC) ^
    (matchUnit_spec r vAddr tblPtrADDR match) ^
    (muxUnit_spec r vAddr latOut rAddr xlat) ^
    (memoryUnit_spec r rReq rAddr data fdone mem))

```

The implementation definition connects the datapath with the control unit. The state consists of the table pointer register value, the security Data register and the control unit phase (tblPtr, secData, phase). The input environment is provided by the system bus and the memory (vAddr, vData, rwe, superv, reqIn, mem). The output environment includes a real address and several control unit outputs (rAddr, done, ack, xlat). The memory address of the table pointer register is specified by the constant tblPtrADDR.

#### Correctness Statement

Several auxiliary definitions are used to express the final correctness statement. To relate the implementation to the specification, a temporal abstraction is constructed using the two predicates Next and First[9]. The predicate First is true when its argument  $t$  is the first time that  $g$  is true. The predicate Next is true when  $t2$  is the next time after  $t1$



---

```

 $\vdash_{def}$  controlUnit_spec reqIn super rwe match secOK fdone muxC tmpC tblC
      lC rReq xlat done ack phase =
      ((muxC 0,tmpC 0,tblC 0,lC 0,rReq 0,xlat 0,done 0,ack 0,phase
0)=(0,F,F,F,F,F,F,F,0))
       $\wedge$ 
      ( $\forall t$  . (muxC(t+1),tmpC(t+1),tblC(t+1),lC(t+1),rReq(t+1),xlat(t+1),done(t+1),
      ack(t+1),phase(t+1)) =
      (phase t = 0)  $\rightarrow$  (reqIn t  $\rightarrow$ 
      (phase t = 1)  $\rightarrow$  (super t  $\rightarrow$ 
      ((wBIT (rwe t))  $\wedge$  match t)  $\rightarrow$ 
      ((phase t = 2)  $\wedge$  fdone t)  $\rightarrow$ 
      ((phase t = 3)  $\wedge$  fdone t)  $\rightarrow$  (secOK t  $\rightarrow$ 
      (phase t = 4)  $\rightarrow$ 
      (phase t = 5)  $\rightarrow$ 
      (muxC t,tmpC t,tblC t,lC t, F ,xlat t,done t,ack t,phase t)))
      % M t t l r x d a P %
      % U m b a e l o c H %
      % X p l t q t n k A %
      ( 0, F,F,F, F,F,F,F, 1) |
      ( 0, F,F,F, F,F,F,F, 0)) |
      ( 0, F,T,F, F,F,F,F, 5) |
      ( 0, F,F,F, F,F,T,T, 0) |
      ( 2, T,F,T, T,T,F,F, 2)) |
      ( 1, F,F,F, T,T,F,F, 3) |
      ( 0, F,F,F, F,T,F,F, 4) |
      ( 0, F,F,F, F,F,T,F, 0)) |
      ( 0, F,F,T, F,T,T,T, 0) |
      ( 0, F,F,F, F,F,T,T, 0) |

```

---

that  $g$  is true. The predicate `stable_sigs` states that between  $t1$  and  $t2$  the MMU inputs will remain constant.

---

```

 $\vdash_{def}$  First g t = ( $\forall p$ :time.  $p < t \Rightarrow \neg(g p)$ )  $\wedge$  (g t)
 $\vdash_{def}$  Next g (t1,t2) = (t1 < t2)  $\wedge$ 
      ( $\forall t$ :time . t1 < t  $\wedge$  t < t2  $\Rightarrow \neg(g t)$ )  $\wedge$  (g t2)
 $\vdash_{def}$  stable_sigs t1 t2 vAddr rwe tblPtrADDR data
      mem super =  $\forall t'$ . t1 < t'  $\wedge$  t' < t2  $\Rightarrow$ 
      (super t' = super t1)  $\wedge$  (vAddr t' = vAddr t1)  $\wedge$  (rwe t' = rwe t1)  $\wedge$ 
      (data t' = data t1)  $\wedge$  (tblPtrADDR t' = tblPtrADDR t1)  $\wedge$  (mem t' = mem t1)

```

---

The correctness theorem states that if the implementation is in phase 0 and a memory request is made, the implementation will eventually respond ( $c$  time steps later), when the state of the implementation matches the state defined by the specification for a set of given MMU inputs. The inputs must remain stable until the MMU responds to a request. If a memory request is not made, the acknowledgment line remains F, the phase remains 0 and the MMU table pointer register remains unchanged.

---

```

 $\vdash$  mmu_imp r vAddr vData rwe super tblPtr tblPtrADDR
      reqIn rAddr done ack xlat mem phase  $\Rightarrow$ 
      ( $\forall t$ . (phase t = 0)  $\Rightarrow$ 
      (reqIn t  $\rightarrow$ 
      ( $\exists c$ . Next done(t,t + c)  $\wedge$  (phase(t + c)=0)  $\wedge$ 
      (stable_sigs t (t + c) vAddr rwe tblPtrADDR
      vData mem super  $\Rightarrow$ 
      (mmu_spec r (vAddr t) (rwe t) (tblPtrADDR t)
      (tblPtr t) (vData t) (mem t) (super t)
      = ack(t + c),rAddr(t + c),tblPtr(t + c))))
      | ( (ack(t + 1) = F)  $\wedge$ 
      (phase(t + 1) = 0)  $\wedge$ 
      (tblPtr(t + 1) = tblPtr t) ) ))

```

---

### 3 Memory Subsystem

An initial design integrated a FIFO cache stack inside the MMU but here we model a fully associative cache as part of the memory subsystem. The cache is described as a lookup table and implements a least recently used (LRU) replacement strategy. Each table entry consists of a key, a related data word, and a boolean indicating whether the entry is active. We will first describe the specification of the LRU replacement strategy in HOL, followed by the cache implementation.

---

```
TAB_ENTRY = ":bool#*address#*wordn" : type
TAB       = ":(~TAB_ENTRY)list" : type
```

---

```
⊢def live entry    = (FST entry)
⊢def key entry     = (FST (SND entry))
⊢def content entry = (SND (SND entry))
```

---

Several auxiliary (recursive) definitions describe table operations below. When an entry is inserted into the top of table, the entry at the bottom will be lost only when the table is "full" (all entries are live). In this respect, the table acts as a queue.

---

```
⊢def (TAB_FULL tbl 0 = live (EL 0 tbl)) ∧
    (TAB_FULL tbl (SUC n) = (live (EL (SUC n) tbl) ∧ TAB_FULL tbl n))
⊢def (TAB_INSERT tbl entry 0 = [entry]) ∧
    (TAB_INSERT tbl entry (SUC n) = (APPEND (TAB_INSERT tbl entry n)
      ((TAB_FULL tbl n) → [(EL n tbl)] | [(EL (SUC n) tbl)])) )
```

---

A table lookup is successful if there is a key match for one of the entries. For a table size of  $n$ , TAB\_HIT returns (SUC  $n$ ) if the lookup fails.

---

```
⊢def KEY_MATCH rep tbl sg *address n =
    (live(EL n tbl) ∧ ((addrEq rep) (key(EL n tbl), sg)) )
⊢def (TAB_HIT rep tbl sg m 0 =
    ((KEY_MATCH rep tbl sg 0) → 0 | (SUC m))) ∧
    (TAB_HIT rep tbl sg m (SUC n) =
    ((KEY_MATCH rep tbl sg (SUC n)) → (SUC n) | TAB_HIT rep tbl sg m n))
```

---

Frequently, a single matched entry must be invalidated. This can occur due to the LRU policy or a memory write operation. Occasionally, the entire cache must be invalidated at the request of the operating system. The LRU policy requires that if a key match occurs, the entry be inserted at the top of the table. By invalidating the matched entry before the insertion, a table overflow will not occur. LRU\_LOOKUP returns the requested data value and the updated cache table.

---

```

 $\vdash_{def}$  ENTRY_INVALIDATE entry = (F ,key entry, content entry)
 $\vdash_{def}$  (TAB_INVALIDATE tbl 0 = [(ENTRY_INVALIDATE (EL 0 tbl))] )  $\wedge$ 
    (TAB_INVALIDATE tbl (SUC n) =
        (APPEND (TAB_INVALIDATE tbl n) [(ENTRY_INVALIDATE (EL (SUC n) tbl))]))
 $\vdash_{def}$  (DEL_TAB_ENTRY rep tbl sg 0 =
    ((KEY_MATCH rep tbl sg 0)  $\rightarrow$  [(ENTRY_INVALIDATE (EL 0 tbl))] |
        [(EL 0 tbl)] ))  $\wedge$ 
    (DEL_TAB_ENTRY rep tbl sg (SUC n) =
        (APPEND (DEL_TAB_ENTRY rep tbl sg n)
            ((KEY_MATCH rep tbl sg (SUC n))
                 $\rightarrow$  [(ENTRY_INVALIDATE (EL (SUC n) tbl))]
                | [(EL (SUC n) tbl)] )))

```

---

```

 $\vdash_{def}$  LRU_REPL rep tbl entry n = TAB_INSERT (DEL_TAB_ENTRY rep tbl (key entry) n)
    entry n
 $\vdash_{def}$  LRU_LOOKUP rep mem tbl n addr data newTbl =
    let who = (TAB_HIT rep tbl addr n n) in
    ((who = (SUC n))
         $\rightarrow$  ( data = fetch rep( mem, addr)  $\wedge$ 
            newTbl = TAB_INSERT tbl (T,addr,(fetch rep(mem,addr) )) n )
        | (data = (content (EL who tbl)  $\wedge$ 
            newTbl = LRU_REPL rep tbl (EL who tbl) n)

```

---

Using the above definitions, the cache-memory subsystem can be defined. This definition replaces `memoryUnit_spec` in the MMU specification and the new system is verified in a similar manner. The proof shows that the cache/memory system is consistent with the MMU memory model requirements.

---

```

 $\vdash_{def}$  cache_mem_spec r req rwe addr data done mem tbl n =
    (done 0 = F)  $\wedge$ 
    ( $\forall$  t. (req t)  $\rightarrow$ 
        ( $\exists$  t'. Next done (t, t+t')  $\wedge$ 
            (wBIT (rwe t)  $\Rightarrow$ 
                ( (mem (t+t') = store r (mem t,addr t,data t) )  $\wedge$ 
                    (tbl (t+t') = DEL_TAB_ENTRY r (tbl t) (addr t) n ) |
                    ( LRU_LOOKUP r (mem t) (tbl t) n (addr t)
                        (data (t+t')) (tbl (t+t')) )  $\wedge$ 
                        (mem (t+t') = mem t) ) ) )
        |
            ( (done (t+1) = F)  $\wedge$ 
                (mem (t+1) = mem t)  $\wedge$ 
                (tbl (t+1) = tbl t) ) )

```

---

### Cache Implementation

The cache implementation consists of a control unit and a stack of cache cells. Cache cells are the instantiation of the table entries described above—their state consisting of

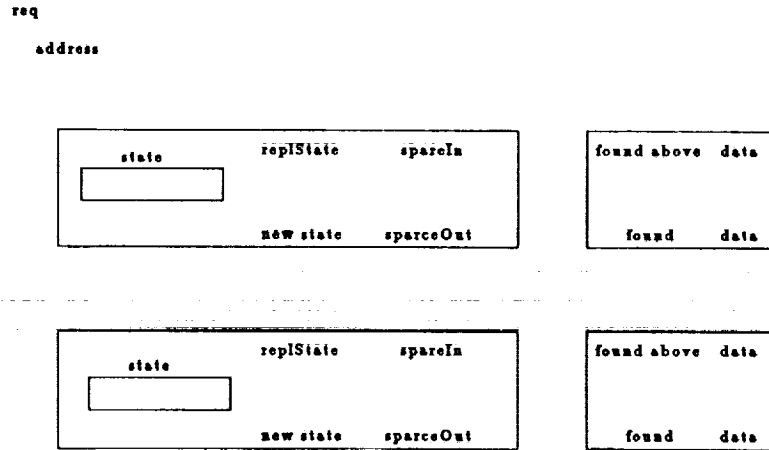


Figure 2: Cache Cell Stack

the three tuple: (valid, address key, data). The action of each cache cell is defined by a two bit function code (*req*) sent by the cache control unit. The stack is formed by joining the outputs of a cache unit to the inputs of the next.

---

```

 $\vdash_{def}$  (cache_block r state req sparseIn foundIn addr replState dataIn 0 =
    cache_cell rep 0 req addr replState (state,sparseIn,foundIn,dataIn))
     $\wedge$ 
    (cache_block rep state req sparseIn foundIn addr replState dataIn (SUC n) =
        (cache_cell rep (SUC n) req addr (EL n state)
            (cache_block rep state req sparseIn foundIn addr replState dataIn n)))

```

---

```

 $\vdash_{def}$  cache_cell_spec rep n req addr replState (stateIn,sparseIn,foundIn,dataIn) =
    let state = (EL n stateIn) in
    let match = ( addrEq rep(addr,key state)  $\wedge$  live state ) in
    (req = (F,F))  $\rightarrow$  % IDLE %
        (stateIn, foundIn, (sparseIn  $\vee$  ~live state), dataIn ) |
    (req = (F,T))  $\rightarrow$  % INVALIDATE ON MATCH %
        ( match  $\rightarrow$ 
            (SET_EL n stateIn(F,key state,content state), T, T, content state ) |
            (stateIn, foundIn, (sparseIn  $\vee$  ~live state), dataIn ) ) |
    (req = (T,F))  $\rightarrow$  % INVALIDATE %
        (SET_EL n stateIn(F,key state,content state), foundIn, T, dataIn ) |
    %req = (T,T)  $\rightarrow$  PUSH DOWN %
        ( sparseIn  $\rightarrow$ 
            (stateIn, foundIn, T, content state )
            (SET_EL n stateIn replState, foundIn, F, dataIn ) )

```

---

When a memory request is made, the control unit signals each cache cell to invalidate its entry if its key matches the input address (F,T). Memory write requests are also passed through to memory. If a read request is pending and the value is not in the cache, the value is fetched from memory. We assume one clock cycle is needed to read a value out of the cache if it is available. After the value fetch step is completed, the control unit pushes the new value onto the cache cell stack by issuing request (T,T).

To model memory, the cache implementation uses the same memory unit specification (`memoryUnit_spec`) stated previously. We then verify that the implementation behaves as specified. The implementation also provides a means of invalidating the entire table (`request (T,F)`), however, this function is not present currently in the specification.

## 4 Summary

We have described the formal verification of an MMU and cache/memory subsystem. The MMU has been verified to perform correctly with an asynchronous memory model. The cache specification defines an LRU replacement policy which is implemented by an electronic block level design. The cache is also demonstrated to be consistent with MMU memory model requirements.

It has been convenient to represent the behavior of devices using abstract representations. This mechanism allows the verification effort to focus on the correctness of higher level abstraction. To verify a more concrete implementation, the abstract representation can be instantiated with components that implement concrete behavior. Extending this example, we plan to demonstrate how a complete system composed of many devices can be shown to correctly implement an abstract system specification.

## References

- [1] W. R. Bevier, "Kit and the Short Stack," *Journal of Automated Reasoning*, vol. 5, 1989.
- [2] M. Gordon, "HOL: A Proof Generating System for Higher-Order Logic," in *VLSI Specification, Verification, and Synthesis*, (G. Birtwhistle and P. Subrahmanyam, eds.), Kluwer Academic Press, 1988.
- [3] E. T. Schubert and K. N. Levitt, "Verification of Memory Management Units," *2nd IFIP Working Conference on Dependable Computing for Critical Applications*, February 1991.
- [4] E. T. Schubert, "Formal Verification of an LRU Cache in Higher Order Logic," technical report CSE-91-, University of California, Davis, September 1991.
- [5] W. R. Bevier, W. A. Hunt, and W. D. Young, "Toward Verified Execution Environments," *IEEE Symposium on Security and Privacy*, 1987.
- [6] A. Cohn, "A Proof of Correctness of the VIPER Microprocessor: the First Level," in *VLSI Specification, Verification, and Synthesis*, (G. Birtwhistle and P. Subrahmanyam, eds.), Kluwer Academic Press, 1988.
- [7] W. A. Hunt, "A Verified Microprocessor," Technical Report 47, The University of Texas at Austin, Dec. 1985.

- [8] W. A. Hunt, "Microprocessor Design Verification," *Journal of Automated Reasoning*, vol. 5, 1989.
- [9] J. J. Joyce, *Multi-Level Verification of Microprocessor-Based Systems*. PhD thesis, Cambridge University, December 1989.
- [10] P. J. Windley, *The Formal Verification of Generic Interpreters*. PhD thesis, University of California, Davis, 1990.
- [11] J. J. Joyce, "Totally Verified Systems: Linking Verified Software to Verified Hardware," *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, July 1989.
- [12] E. T. Schubert, "Verification of Memory Management Units using HOL," technical report CSE-90-27, University of California, Davis, August 1990.

## Formal Hardware Verification of Digital Circuits

J. Joyce and C-J. Seger  
Department of Computer Science  
University of British Columbia  
Vancouver, B.C.  
Canada V6T 1W5

**Abstract-** The use of formal methods to verify the correctness of digital circuits is less constrained by the growing complexity of digital circuits than conventional methods based on exhaustive simulation. This paper briefly outlines three main approaches to formal hardware verification: symbolic simulation, state machine analysis, and theorem-proving.

### 1 Introduction

Advances in VLSI fabrication technology have greatly outstripped 'verification capacity' — that is, the capacity of conventional methods for demonstrating that the design of a circuit is correct with respect to a specification of its requirements.

Verification capacity has fallen behind fabrication technology because conventional verification methods do not scale with complexity. These methods are generally based on simulation — they do not scale because the number of simulation cases is likely to increase exponentially if one attempts to maintain the same degree of coverage.

Considerable effort has been made to increase, in a brute-force manner, what coverage can be achieved with simulation. One approach is to distribute the simulation cases over a large number of machines running identical versions of the simulation model. Another brute-force approach has been the development of special-purpose simulation hardware to increase the speed of a simulation by several orders of magnitude. However, these techniques do not offer a satisfactory, long-term solution for verifying digital designs by exhaustive simulation because, in general, the number of simulation cases grows exponentially with the number of components in a design.

Of course, it may be argued that it is not really necessary, for any practical purpose, to exhaustively simulate a design in order to detect every error in a design. Instead, it would be argued that it is only reasonable to simulate the design for a feasible number of representative cases. However, this assumes that there is general-purpose, systematic method for finding a truly representative set of simulation cases. Although one can easily imagine a systematic way of generating some obvious cases, it is clear that digital systems often fail at the "confluence of unrelated or seemingly unrelated events" [25].

Formal methods offer considerable hope for verification techniques which are better able to scale with the complexity of VLSI designs. We can identify three distinct approaches to formal hardware verification, namely, symbolic simulation, state machine analysis, and theorem-proving. These formal approaches to hardware verification are better able to scale

with the complexity of VLSI designs because they exploit powerful tools of mathematics rather than brute-force. A good example is the use of the mathematical induction which is a mainstay of the theorem-proving approach to formal hardware verification. All of these approaches are supported by software tools — many of which have been under constant development for the last decade or longer.

## 2 The Symbolic Simulation Approach

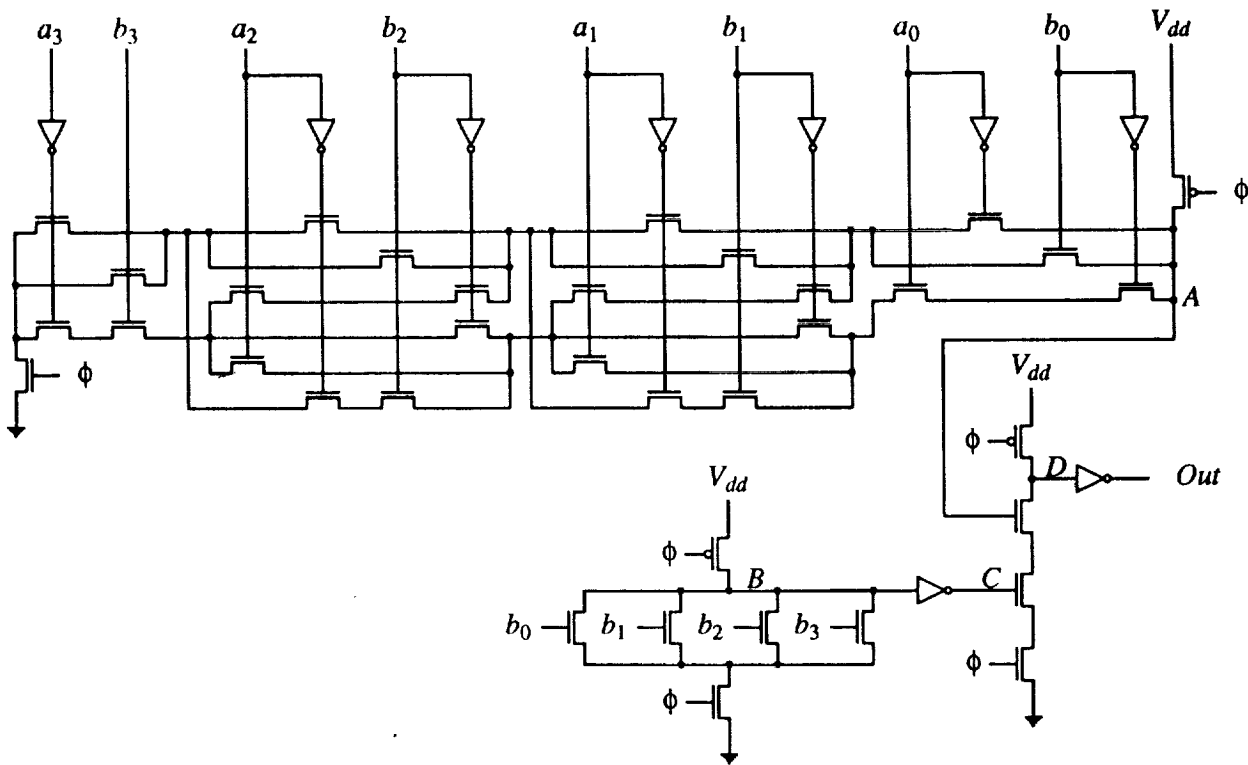
The concept of symbolic simulation was first proposed by researchers in the late 1970's as a method for evaluating register transfer language representations [11]. The early programs were very limited in their analytical power since their symbolic manipulation methods were weak. Consequently, symbolic simulation did not evolve much further until more efficient methods of manipulating symbols emerged. The development of Ordered Binary Decision Diagrams (OBDDs) for representing Boolean functions [8] radically transformed symbolic simulation.

The first "post-OBDD" symbolic simulators were simple extensions of traditional logic simulators [7]. In these symbolic simulators the input values could be arbitrary Boolean expressions over some Boolean variables rather than only 0's, 1's (and possibly X's) as in traditional logic simulators. Consequently, the results of the simulation were not single values but rather Boolean functions describing the behavior of the circuit for the set of all possible data represented by the Boolean variables. To illustrate this idea, consider the (pseudo) Domino CMOS circuit shown in Fig. 1. If the circuit is clocked correctly, the inputs are stable long enough before the clock goes high, and the inputs and clock signal are then kept stable, the output node should eventually change to 1 if and only if the number represented by the 4-bit binary input vector  $a$  is greater than the number represented by the 4-bit binary input vector  $b$ , and both numbers are greater than zero. In a simple OBDD-based symbolic simulator we would simply apply the Boolean input variables at the correct time and in the end compare the value on the output node with the Boolean function:

$$(a_3\bar{b}_3 + (\overline{a_3 \oplus b_3})(a_2\bar{b}_2 + (\overline{a_2 \oplus b_2})(a_1\bar{b}_1 + (\overline{a_1 \oplus b_1})a_0\bar{b}_0)))(b_3 + b_2 + b_1 + b_0)$$

A verifier based on symbolic simulation applies logic simulation to compute the circuit's response to a series of stimuli chosen to detect all possible design errors. When a circuit has been "verified" by simulation, this means that any further simulation would not uncover any errors. Hence, the problem of verifying the correctness of a design becomes one of simulating a large number of input patterns. Selecting such a set of simulation patterns is a nontrivial task, since errors that arise during the design process cannot be easily characterized. Designer's misconceptions, incomplete or inconsistent specifications, and carelessness on the part of the designer can cause the resulting circuit to behave unpredictably. Worst of all, it may misbehave only under unusual combinations of circumstances. Rather than trying to postulate a simplified "fault model" for design errors, it is more appropriate to



Figure 1: Circuit for computing  $A > B > 0$ .

adopt a philosophy that design verification must work against a malicious adversary. That is, given a proposed simulation test, the adversary will attempt to create a circuit that does not fulfill the specification, yet passes the test. A circuit is considered "correct" only if no adversary can defeat the simulation test. Thus, when a circuit has been "verified" by simulation, this means that any further simulation would not uncover any errors.

Since a symbolic simulator is based on a traditional logic simulator, it can use the same, quite accurate, electrical and timing models to compute the circuit behavior. For example, a detailed switch-level model, capturing charge sharing and subtle strengths phenomena, and a timing model, capturing bounded delay assumptions, are well within reach. Also—and of great significance—the switch-level circuit used in the simulator can be extracted automatically from the physical layout of the circuit. Hence, the correctness results will link the physical layout with some higher level of specification.

Recently, Bryant and Seger [10] developed a new generation of symbolic simulator based verifier. Here the simulator establishes the validity of formulas expressed in a very limited, but precisely defined, temporal logic. This temporal logic allows the user to express properties of the circuit over *trajectories*: bounded-length sequences of circuit states. The verifier checks the validity of these formulas by a modified form of symbolic simulation. Further, by exploiting the 3-valued modeling capability of the simulator, where the third logic value  $X$  indicates an unknown or indeterminate value, the complexity of the symbolic

manipulations is reduced considerably.

This verifier supports a verification methodology in which the desired behavior of the circuit is specified in terms of a set of assertions, each describing how a circuit operation modifies some component of the (finite) state or output. The temporal logic allows the user to define such interface details as the clocking methodology and the timing of input and output signals. The combination of timing and state transition information is expressed by an assertion over state trajectories giving properties the circuit state and output should obey at certain times whenever the state and inputs obey some constraints at earlier times.

This form of specification works very well for circuits that are normally viewed as state transformation systems, i.e., where each operation is viewed as updating the circuit state. Using a prototype system, a simple 32-bits microprocessor and a significant portion of a modern 32 bit RISC microprocessor have been verified. These circuits contained around 15,000 transistors and the verification effort required less than two hours on a MIPS Magnum 3000 workstation. The complete verification process including developing the specification, deriving the circuit description, and carrying out the symbolic ternary simulation, took less than a person-week.

### 3 State Machine Analysis

A second approach to formal hardware verification is *state machine analysis*. This approach uses algorithmic techniques to decide whether a finite state machine satisfies a set of user-specified properties. In this brief overview, we focus on just one particular approach to state machine analysis called *model-checking*. Other approaches to state machine analysis include those based on *language containment tests* [23].

We use the example of a simple handshaking protocol, illustrated by the timing diagram in Figure 2, to describe the state-machine analysis approach to formal hardware verification.

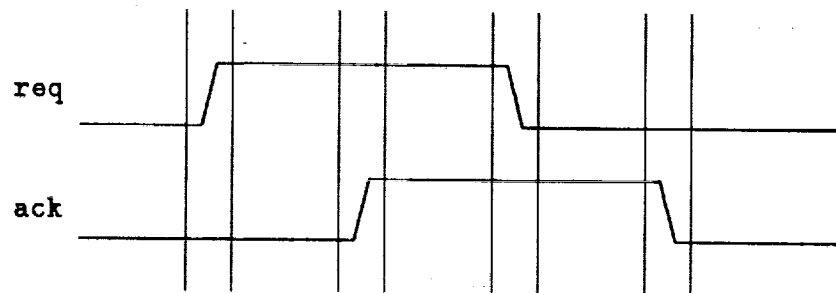


Figure 2: Timing diagram for simple handshaking protocol.

This protocol could be implemented either in software or directly in hardware. A 'correct' implementation of this protocol must satisfy the following properties:

*"whenever the request signal becomes true, it must remain true until it is acknowledged"*

*"every request must eventually be acknowledged"*

*"whenever the acknowledgement signal becomes true, it must remain true until the request signal returns to false"*

*"the request signal will eventually return to false after the request is acknowledged"*

*"whenever the request signal is false, it will remain false until the acknowledgement signal is also false"*

*"the acknowledgement signal will eventually return to false after the request signal returns to false"*

*"once false, the acknowledgement signal will remain false until there is a request"*

*"whenever the acknowledgement signal is false, there will eventually be a request"*

These properties can be translated one-by-one into temporal logic. The symbols  $\cup$ ,  $\Diamond$ ,  $\sim$  and  $\rightarrow$  can be informally read as "until", "eventually", "not" and "implies".

$$(\text{req} \rightarrow (\text{req} \cup \text{ack}))$$

$$(\text{req} \rightarrow (\Diamond \text{ack}))$$

$$(\text{ack} \rightarrow (\text{ack} \cup (\sim \text{req})))$$

$$(\text{ack} \rightarrow (\Diamond (\sim \text{req})))$$

$$((\sim \text{req}) \rightarrow ((\sim \text{req}) \cup (\sim \text{ack})))$$

$$((\sim \text{req}) \rightarrow (\Diamond (\sim \text{ack})))$$

$$((\sim \text{ack}) \rightarrow ((\sim \text{ack}) \cup \text{req}))$$

$$((\sim \text{ack}) \rightarrow (\Diamond \text{req}))$$

A program for automatic state machine analysis would take, as input, a machine-readable list of formally specified properties such as the eight properties listed above. The analyzer program would also take, as input, a machine-readable description of a finite state machine, for example, a model of a candidate implementation of the handshaking protocol. The analyzer program would then generate either the answer "yes", meaning that the state machine does indeed satisfy all of the properties supplied by the user, or the answer "no", meaning that the machine fails to satisfy at least one of these properties. When the outcome is "no", the analyzer may also produce helpful information about how the state machine fails to satisfy a particular property, i.e., a counter-example.

State-machine analysis is bounded by the number of states in the finite state machine. Early state machine analysis techniques relied on the explicit enumeration of states which,

reportedly, limits the use of these techniques to systems with between  $10^3$  and  $10^6$  reachable states. Unfortunately, the number of states in a system may grow exponentially with the number of concurrent components in the system. To deal with this “state explosion problem”, several groups [3,9,16] have investigated ways to represent a state space symbolically rather than explicitly. A popular candidate for the symbolic representation of states are OBDD’s — mentioned earlier in connection with the symbolic simulation approach. Using this symbolic approach, it is reported that state machine analysis can be applied in practice to systems with in excess of  $10^{20}$  states [9].

State machine analysis techniques have been applied to several commercial designs. These techniques were used to discover several possible execution sequences leading to failure in a design for the cache consistency protocol of the Encore Gigamax multiprocessor [24]. Another approach to state machine analysis (based on language containment) has been used by AT&T in the design of a packet layer controller chip [23].

## 4 Theorem Proving

A third approach to formal hardware verification, computer-assisted *theorem-proving*, is based on the construction of a proof in formal logic. This proof is a formal argument that a hardware design, based on some model of the primitive components, satisfies a formal specification of its requirements. Figure 3 shows an example of a formal proof establishing the correctness of the two-component design shown in Figure 4.

1. <code>ANDGate_IMP (i1,i2,outh)</code>	[from above circuit diagram]
2. $\exists x. \text{NANDGate } (i1,i2,x) \wedge \text{NOTGate } (x,outh)$	[by def. of <code>ANDGate_IMP</code> ]
3. <code>NANDGate (i1,i2,x) <math>\wedge</math> NOTGate (x,outh)</code>	[strip off “ $\exists x.$ ”]
4. <code>NANDGate (i1,i2,x)</code>	[left conjunct of line 3]
5. $x = \neg(i1 \wedge i2)$	[by def. of <code>NANDGate</code> ]
6. <code>NOTGate (x,outh)</code>	[right conjunct of line 3]
7. $outh = \neg x$	[by def. of <code>NOTGate</code> ]
8. $outh = \neg(\neg(i1 \wedge i2))$	[substitution, line 5 into 7]
9. $outh = (i1 \wedge i2)$	[simplify, $\neg\neg t = t$ ]
10. <code>ANDGate (i1,i2,outh)</code>	[by def. of <code>ANDGate</code> ]
11. <code>ANDGate_IMP (i1,i2,outh)</code> $\implies \text{ANDGate } (i1,i2,outh)$	[discharge assumption, line 1]

Figure 3: Formal proof of correctness for an AND-Gate.



Figure 4: Implementation of an AND-gate from an NAND-gate and an inverter.

Both ordinary human reasoning and formal proof can be used to show that a specific conclusion follows from a certain set of assumptions by accepted laws of reasoning. However, formal proof is a purely syntactic process. A proof is formally defined as a sequence of lines (such as the numbered sequence of lines in Figure 3) where each line follows from a previous line by rule of inference. There are only a finite number of primitive inference rules (in fact, usually a very small number of primitive rules). The validity of any particular line in a proof can be decided by a purely syntactic test based on checking to see if any one of the primitive inference rules can be used to justify that particular line in the proof.

Unlike ordinary human reasoning, which is notoriously error-prone, formal proof is extremely rigorous. Indeed, its main advantage is that it can be mechanically checked. The main disadvantage of formal proof, compared to ordinary human reasoning, is that formal proof is overwhelmingly tedious. The very simple proof in Figure 3 has just eleven lines, but a formal proof of correctness for a real design (such as a simple microprocessor) may involve several million primitive inference steps. Fortunately, there has been considerable progress made towards the partial automation of formal proof. A very large fraction of the actual line-by-line inference steps in a formal proof can be generated automatically by computer-based theorem-prover.

A digital circuit can be “verified” using a theorem-prover by generating a theorem which states that the formal specification of a design logically *satisfies* a formal specification of its intended behaviour (i.e. a high level model). The exact meaning of “satisfies” is stated unambiguously as a mathematical relationship between the two levels of formal specification. In the very simple example shown in Figure 3, logical implication,  $\implies$ , is used to express the relationship between the implementation of the AND-gate and its behavioural specification:

$$\text{ANDGate\_IMP} (i1, i2, \text{outp}) \implies \text{ANDGate} (i1, i2, \text{outp})$$

The theorem-proving approach to formal hardware verification is a *structural* approach in contrast to both symbolic simulation and state-machine analysis which are *behavioural* approaches. The latter two approaches, symbolic simulation and state-machine analysis, both apply verification techniques to a ‘flat’ design — they do not require additional details about the hierarchical structure of the design. On the other hand, theorem-proving can only be applied ‘in the large’ to a hierarchically structured design. In a theorem-proving approach, the design is verified hierarchically. The proof hierarchy is generally a reflection of the hierarchical structure of the design. For example, the bottom level of this hierarchical process may involve the formal verification of simple RTL (Register Transfer

Level) components composed from primitives such as CMOS transistors. Each kind of component only has to be verified once — in contrast to other approaches which verify every instance of that particular component. At higher levels in the verification hierarchy, each instance of a particular component uses the single verification result obtained from a lower level.

The direct re-use of a verification result for multiple identical instances of a particular component is a very simple form of how a single verification result can be re-used. Theorem-proving approaches also allow *generic* specifications to be formally verified where a specification is parameterized by scalar values (e.g., the number of bits in a RTL component) or even by data types and operations [21]. A single generic verification result can be instantiated for different parameter values; for example, the generic specification of an  $n$ -bit multiplier can be instantiated for different values of  $n$ , e.g., a 16-bit multiplier, a 32-bit multiplier.

The theorem-proving approach relies heavily upon (and benefits greatly from) a number of mathematical tools. This includes, as with symbolic simulation, the ability to represent data symbolically. Mathematical induction is also critical for scaling with the increasingly complexity of circuit designs.

A distinct advantage of the theorem-proving approach to formal hardware verification is the ability to verify digital systems with respect to higher algebraic levels. For example, the correctness of arithmetic hardware can be stated directly in terms of arithmetic operations on natural numbers rather than Boolean operations on bit-vectors. This is often referred to as *data abstraction* — an illustrative example of this technique is given by Chin [12] in verifying arithmetic hardware for signal processing applications. Other kinds of abstraction include *temporal abstraction* which is a technique for relating computational behaviour at increasingly abstract time scales.

Among the best known interactive theorem-provers are the Boyer-Moore Theorem Prover [4] and the Cambridge HOL System [18,19]. The Boyer-Moore Theorem Prover has been used by researchers at Computational Logic Inc. to develop an multi-level proof of correctness for a complete computer system including both hardware and software levels [1,2]. The Cambridge HOL System has been used by researchers at Cambridge University to verify aspects of the commercially-available Viper microprocessor designed by the British Ministry of Defence for safety-critical applications [6,13,14,15].

## 5 Summary and Future Directions

This paper has briefly described three main approaches to formal hardware verification: symbolic simulation, state-machine analysis and theorem-proving. There have already been some trial applications of these verification techniques to real commercial designs (as mentioned earlier) and there is evidence of increasing industrial interest in these techniques.

Many research efforts in this area are now focussed on the issue of integrating formal verification techniques with conventional CAD. For example, researchers at Cambridge University are investigating the use of conventional HDL's (Hardware Description Lan-

guages) such as Ella and VHDL as specification languages for theorem-proving techniques [5]. Another example is work that investigates links between formally verified hardware specifications and conventional CAD tools such as silicon compilers [22].

It is unlikely that any one of the three verification approaches described in this paper offers, by itself, a "complete" approach to verifying digital hardware. However, we believe that a "complete" approach may be achieved by some combination of the three approaches described here. We are currently developing a hybrid approach that based on a combination of symbolic simulation (using the COSMOS system) and theorem-proving (using the Cambridge HOL system). The objective of our research is a hybrid formal verification methodology (and supporting tools) which combines the complementary advantages of theorem-proving and symbolic simulation. This methodology would allow a very abstract specification of a digital system (specified with the full expressive power of higher-order logic) to be verified with respect to a switch-level model of a CMOS digital circuit. Initial progress on the development of a "mathematical interface" for this hybrid approach is reported in [26].

## Acknowledgements

This research is supported by Operating Grants from the Natural Science and Engineering Research Council of Canada (NSERC).

## References

- [1] W. R. Bevier, W. A. Hunt, Jr., and W. D. Young, Towards Verified Execution Environments, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 27-29 April 1987, Oakland, California Computer Society Press, Washington, D.C., 1987 pp. 106-115. Also Report No. 5, Computational Logic, Inc., Austin, Texas, February 1987.
- [2] W. Bevier, W. Hunt, J Moore, and W. Young, An Approach to Systems Verification, *Journal of Automated Reasoning*, Vol. 5, No. 4, November 1989. Also Report No. 41, Computational Logic, Inc., Austin, Texas, April 1989.
- [3] S. Bose and A. Fisher, Automatic Verification of Synchronous Circuits using Symbolic Logic Simulation and Temporal Logic, *Proceedings of the IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Designs*, Houthalen, Belgium, 1989.
- [4] R. S. Boyer and J S. Moore, *A Computational Logic*, Academic Press, 1979.
- [5] R. Boulton, M. Gordon, J. Herbert, and J. Van Tassel, The HOL Verification of ELLA Designs, *Proceedings of the ACM 1991 International Workshop on Formal Methods in VLSI Design*, P. Subrahmanyam, ed., Miami, Florida, 9-11 January 1991
- [6] B. Brock and W. A. Hunt, Jr., Report on the Formal Specification and Partial Verification of the VIPER Microprocessor, Report No. 46, Computational Logic, Inc., Austin, Texas, January 1990.

- [7] R.E. Bryant, Symbolic Verification of MOS Circuits. *1985 Chapel Hill Conference on VLSI*, Fuchs, H., Ed. Computer Science Press, Rockville, MD, 1985, pp. 419-438.
- [8] R.E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, No. 8, December 1986, pp. 677-691.
- [9] J. Burch, E. Clarke, and K. McMillan, Symbolic Model Checking:  $10^{20}$  States and Beyond. Manuscript, 1990.
- [10] R.E. Bryant and C-J. Seger, Formal Verification of Digital Circuits Using Symbolic Ternary System Models, *Computer-Aided Verification '90*, eds. E.M. Clarke, R.P. Kurshan, American Mathematical
- [11] W.C. Carter, W.H. Joyner, Jr., and D. Brand, Symbolic Simulation for Correct Machine Design, *16th ACM/IEEE Design Automation Conference*, 1979, pp. 280-286.
- [12] S-K. Chin, Synthesis of Arithmetic Hardware Using Hardware Metafunctions, *IEEE Trans. CAD*, Vol. 9, No. 8, August 1990, pp. 793-803.
- [13] Avra Cohn, A Proof of Correctness of the Viper Microprocessor: The First Level, *VLSI Specification, Verification and Synthesis*, G. Birtwistle and P. Subrahmanyam, eds., Kluwer Academic Publishers, Boston, 1988, pp. 27-71. Also Report No. 104, Computer Laboratory, Cambridge University, January 1987.
- [14] Avra Cohn, Correctness Properties of the Viper Block Model: The Second Level, *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. Subrahmanyam, eds., Springer-Verlag, 1989, pp. 1-91. Also Report No. 134, Computer Laboratory, Cambridge University, May 1988.
- [15] Avra Cohn, The Notion of Proof in Hardware Verification, *Journal of Automated Reasoning*, Vol. 5, May 1989, pp. 127-139.
- [16] O. Coudert, C. Berthet, and J. C. Madre. Verification of Synchronous Sequential Machines based on Symbolic Execution, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.
- [17] M. J. C. Gordon, Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware, *Formal Aspects of VLSI Design, Proceedings of the 1985 Edinburgh Conference on VLSI*, G. Milne and P. Subrahmanyam, eds., North-Holland, 1986, pp. 153-177.
- [18] M.J.C. Gordon, Mechanizing Programming Logics in Higher Order Logic, *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. Subrahmanyam, eds., Springer-Verlag, 1989, pp. 387-439. Also Report No. 145, Computer Laboratory, Cambridge University, September 1988.
- [19] M.J.C. Gordon et al., The HOL System Description, Cambridge Research Centre, SRI International, Suite 23, Miller's Yard, Cambridge CB2 1RQ, England.
- [20] Warren A. Hunt, FM8501, A Verified Microprocessor, Ph.D. Thesis, Report No. 47, Institute for Computing Science, University of Texas, Austin, December 1985.
- [21] Jeffrey J. Joyce, Generic Specification of Digital Hardware, *Proceedings of Workshop on Digital Circuit Correctness*, M. Sheeran and G. Jones, eds., September 1990, Oxford.



- [22] J. Joyce, E. Liu, J. Rushby, N. Shankar, R. Suaya, and F. von Henke, From Formal Verification to Silicon Compilation, Proceedings of *COMPCON 91*, San Francisco, California, 26-27 February 1991.
- [23] R. Kurshan, Automaton (State-Machine)-Based Analysis, *Tutorial on Formal Verification of Hardware*, DAC'91, 21 June 1991, San Francisco.
- [24] K. McMillan and J. Schwalbe, Formal Verification of the Gigamax Cache Consistency Protocol, manuscript, 1990.
- [25] P. G. Neuman, The Computer-Related Risk of the Year: Weak Links and Correlated Events, Proceedings of *COMPASS '91*, Washington, D.C., 24-27 June 1991.
- [26] C.-J. Seger and J. J. Joyce, A Two-Level Formal Verification Methodology using HOL and COSMOS, *Proceedings of the Third Workshop on Computer Aided Verification*, K. Larsen and A. Skou, eds., Aalborg, Denmark, 1-4 July 1991. pp. 380-391.



# High Accuracy Switched-Current Circuits Using an Improved Dynamic Mirror

G. Zweigle and T. Fiez

School of Electrical Engineering and Computer Science  
Washington State University  
Pullman, WA 99164

<sup>1</sup> *Abstract* - The switched-current technique, a recently developed circuit approach to analog signal processing, has emerged as an alternative/compliment to the well established switched-capacitor circuit technique. High speed switched-current circuits offer potential cost and power savings over slower switched-capacitor circuits. Accuracy improvements are a primary concern at this stage in the development of the switched-current technique. Use of the dynamic current mirror has produced circuits that are insensitive to transistor matching errors [1]. The dynamic current mirror has been limited by other sources of error including clock-feedthrough and voltage transient errors. In this paper we present an improved switched-current building block using the dynamic current mirror. Utilizing current feedback the errors due to current imbalance in the dynamic current mirror are reduced. Simulations indicate that this feedback can reduce total harmonic distortion by as much as 9dB. Additionally, we have developed a clock-feedthrough reduction scheme for which simulations reveal a potential 10dB total harmonic distortion improvement. The clock-feedthrough reduction scheme also significantly reduces offset errors and allows for cancellation with a constant current source. Experimental results confirm the simulated improvements.

## 1 Introduction

The switched-current (SI) sampled-data signal processing technique is becoming a viable alternative to the switched-capacitor (SC) technique. Unlike SC circuits, which require additional processing steps to fabricate precision linear capacitors, SI circuits can be integrated in a standard digital CMOS process. In addition, SI circuits can operate with low power supply voltages, they can operate at high speeds, and they are very area efficient. The drawback of SI circuits at this time is their limited accuracy. This problem must be overcome in order for switched-current circuits to gain the wide acceptance switched-capacitor circuits have attained. In this paper, an SI circuit is presented that significantly improves the accuracy of the current-mode system.

---

<sup>1</sup>This research was supported in part by a grant from the National Science Foundation Center for the Design of Analog/Digital Integrated Circuits (CDADIC) at Washington State University, University of Washington, and Oregon State University

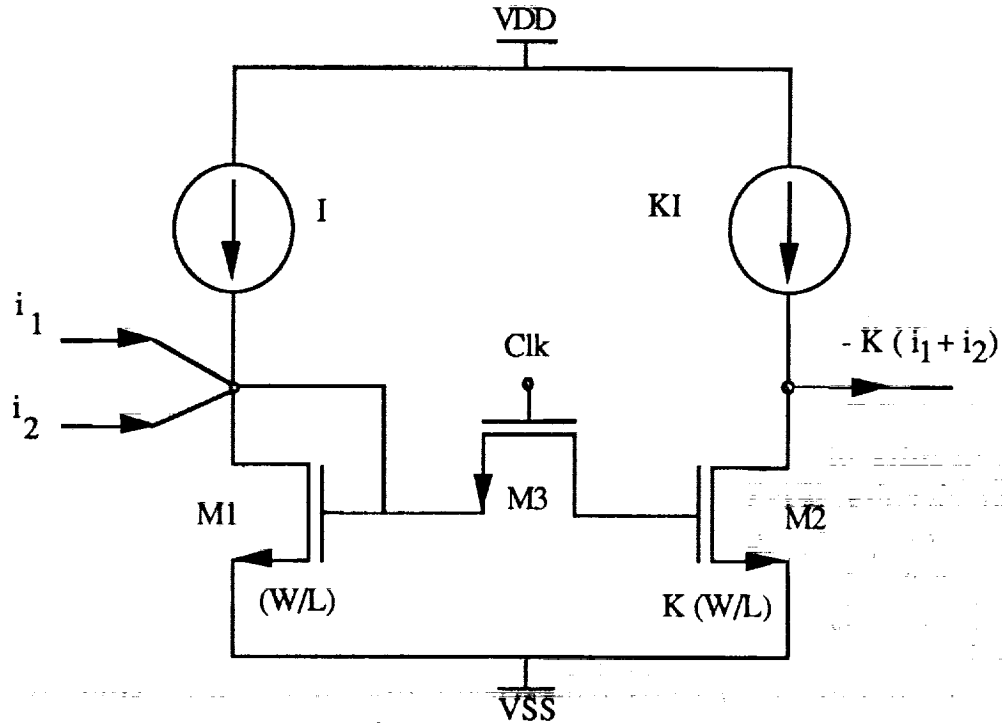


Figure 1: Switched-current track-and-hold circuit.

## 2 Switched-Current Circuit Operation

### 2.1 Current Track-and-Hold

The current track-and-hold (T/H) is a basic building block of switched-current circuits, Fig. 1. Transistors M1 and M2 are biased in saturation by the current sources, indicated as  $I$ , and the track-and-hold operation is controlled by switch transistor M3. When the clock is high, the input current is mirrored to the output. The parasitic gate capacitance of transistor M2 stores a voltage corresponding to the value of the input current. When the clock is low and transistor M3 is turned off, the drain current of M2 is held at a value corresponding to the voltage stored on the gate of M2.

The current track-and-hold performs the four signal processing operations of inversion, summation, scaling, and delay. Consider initially that the clock is high and the gates of M1 and M2 are shorted. When a signal  $i_1$  is input to the diode connected transistor M1, it is mirrored to transistor M2. The drain current of M2 is  $I + i_1$ . The output current is  $-i_1$ . This stage inverts the current. The output current is a sum of two input currents by simply connecting wires. Scaled current output is obtained by scaling the aspect ratio of M2 to M1. Finally, signal delay is controlled by switching transistor M3 on and off.

By using these basic signal processing operations, current track-and-hold circuits can be combined to perform more complicated operations. One of these, the integrator, is realized by cascading two current track-and-holds with feedback as shown in Figure 2.

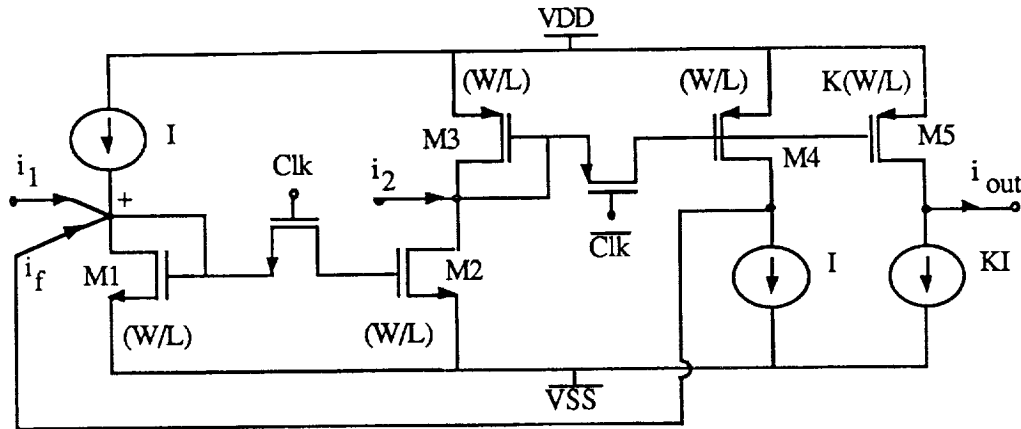


Figure 2: A switched-current differential integrator.

The transfer function of this circuit is

$$i_{out}(z) = \frac{(K(i_1 z^{-1} - i_2 z^{-0.5}))}{(1 - z^{-1})}. \quad (1)$$

The non-inverting integrator input is at the input of the first T/H and the inverting integrator input is at the input of the second T/H. The switched-current integrator has been shown to be directly analogous to the switched-capacitor integrator [2]. Note that, as with the SC integrator, the two switches of the SI integrator are controlled by two phase non-overlapping clocks. Additionally, the integrator coefficient  $K$  is determined by the aspect ratio of transistor  $M5$  to transistor  $M3$ . In the SC integrator a capacitor ratio determines this factor. The reliance of this switched-current circuit on transistor matching has contributed to its limited accuracy.

## 2.2 Dynamic Current Mirror

The dynamic current mirror eliminates matching errors present in simple current track-and-holds by mirroring current in time rather than space, Figure 3. Operation of the dynamic mirror is controlled by switches  $MC1$ ,  $MC2$ , and  $MC3$ . These switches require a two phase non-overlapping clock, similar to the current track-and-hold integrator. It has been shown that an integrator composed of the dynamic mirror cell does

not require any more clocks than the SI integrator presented previously [3]. Transistor  $M1$  is biased by the DC current source  $I$ . Initially the switches  $MC1$  and  $MC2$  are closed. The signal current is read into the diode connected transistor  $M1$ , producing a voltage on its gate. This voltage is proportional to the square root of the input current for saturated operation. The current is then read out by opening switches  $MC1$  and  $MC2$  while switch  $MC3$  is closed. The stored voltage produces an output current that is an inverted replica of the input current. The dynamic mirror differs from the simple track-and-hold by using switches to time multiplex one transistor, resulting in a float-and-hold operation. Unlike the simple current mirror track-and-hold, where the switch transistor passes nearly zero current, the controlling switches  $MC2$  and  $MC3$  of the dynamic mirror must pass the signal

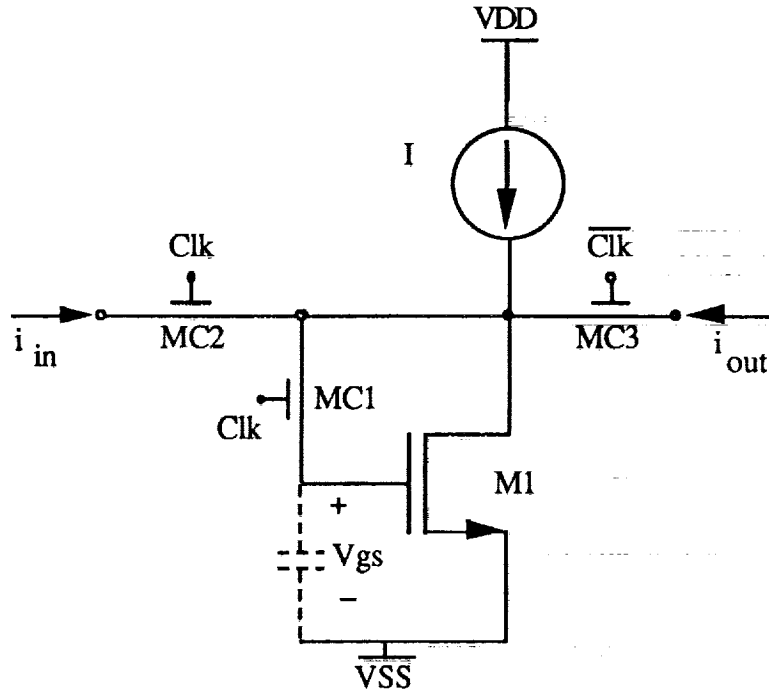


Figure 3: Dynamic current mirror float-and-hold cell.

current. Also, since the same transistor is used to both mirror in and out the signal current, the method of scaling currents by scaling transistor aspect ratios is not possible using the dynamic mirror. To perform signal scaling additional dynamic mirrors are multiplexed in time [4].

### 3 Dynamic Current Mirror Error Sources

Although SI circuits have been shown to be a viable signal processing circuit technique, the poor accuracy limits system performance. Sources of this inaccuracy for the dynamic mirror are finite output impedance effects, clock-feedthrough effects, and voltage spikes. While these effects can be reduced by using large current mirror devices, this solution is not optimum because increasing device size increases area requirements and reduces speed capabilities. The finite output impedance of a dynamic current mirror results in current division between stages. This division of the signal current becomes an AC gain error with magnitude

$$\Delta i = (Z_{in}/Z_{out})i. \quad (2)$$

Ideally the output impedance of the dynamic mirror would be infinite and the input impedance zero. Because these conditions are not met in real implementations, errors are introduced in the output current. The output impedance of a dynamic current mirror can be increased with the use of a cascade circuit. Because of its extremely high output impedance and special feedback properties the regulated gate cascade [5] was employed

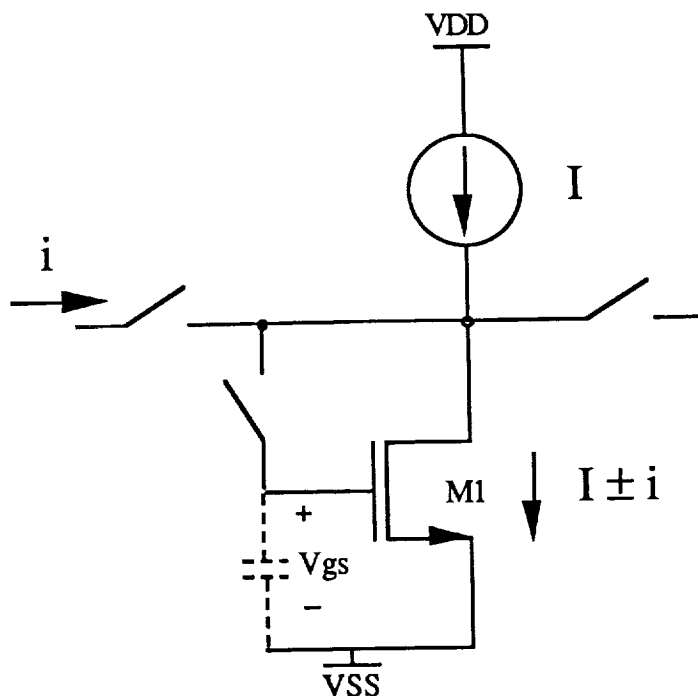


Figure 4: Current mismatch in the dynamic current mirror at the switching interval.

for reducing the AC gain errors. Clock-feedthrough effects in switched-current circuits are more severe than in switched-capacitor circuits [6]. This is because the parasitic gate capacitances of SI circuits are smaller than the linear capacitors implemented in SC circuits. By using larger capacitors to hold the signal, SC circuits reduce the effect of small clock-feedthrough charges. In SI circuits clock-feedthrough results in offset errors and increases total harmonic distortion. Several methods of reducing this injected charge in SI circuits have been studied to date. These include capacitive feedback [7], the use of dummy switches [8], current difference cells [9], and an adaptive clock [7,9]. In this paper a scheme is presented for reducing clock-feedthrough in the dynamic mirror with an improved adaptive clock. Finally, voltage spike errors are introduced in the dynamic mirror by the operation of the two phase nonoverlapping clocks. During the switching of transistors MC2 and MC3 there will be a nonzero interval of time when all of the switches are in the OFF state. During this time period the data holding transistor M1 and the current source will be attempting to draw two different currents. This can be seen in Figure 4. Transistor M1 will have a gate-source voltage set by the input current which will give a drain current of  $I \pm i$ . The current mirror will be delivering current  $I$ . As a result, the voltage at the drain of transistor M1 will have to move to counter the current imbalance. For negative input currents, the voltage will increase in an attempt to shut off the current source and make transistor M1 draw more current. For positive input currents the voltage will decrease, attempting to draw more current from the current source and less from transistor M1.

There are two paths that these voltage transitions can couple through to the data holding node. One is through switch transistor MC1 as it turns off with switch transistor

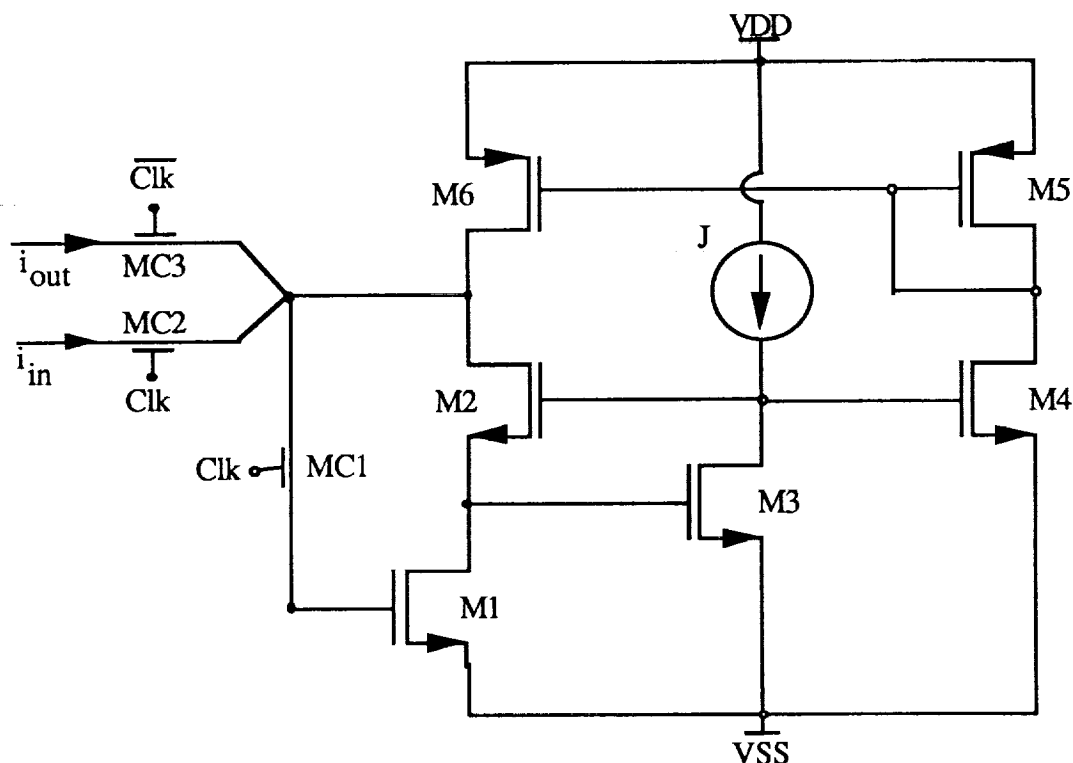
MC2. The other is through the drain-gate capacitance of transistor M1. Although it may seem that a cascade could be used to buffer the drain of transistor M1 from the spikes, this circuit will only be useful when functioning as designed. For positive input currents negative spikes will cause the cascade to leave its proper operating point. Subsequent to this, the drain of M1 will no longer be protected and the spike will couple through  $C_{gs1}$ . The resulting error from the voltage spikes is signal dependent in both magnitude and polarity, difficult to predict, and sometimes worse than switch charge injection effects. The voltage spike error must be eliminated before clock-feedthrough cancellation schemes can be effective.

## 4 Voltage Spike Error Reduction

Two solutions to reducing voltage spike errors have been developed. The first involves modifying the clock phasing of the dynamic mirror. By turning switch transistor MC1 off slightly before turning switch transistor MC2 off, the path of the voltage spike through transistor MC1 is eliminated. The new clocking scheme presented here does not add another clock phase to the circuit, only a delay is needed. Transients occurring when the transistor goes from the hold mode to the output float mode are irrelevant because a new current value will be read into the diode connected transistor at this time. The delay only needs to be as long as the turn off time of transistor MC1. It can be implemented on chip with an even number of cascaded inverters. The other solution to voltage spike errors must eliminate the voltage spike from coupling through the drain-gate capacitance of transistor M1. This is accomplished by using current feedback around a regulated gate cascade dynamic current mirror, Fig. 5. The regulated gate cascade is used to increase the output impedance of the dynamic mirror and the current feedback is used to keep the cascade in its proper operating region for positive input currents. The circuit operates as follows. Transistor M3 senses variations in the drain-source voltage of transistor M1. Since a constant current biases M3, these variations will be amplified by the loop gain of transistors M3 and follower M2. Differences between the drain-source voltage of transistor M1 and the gate voltage of transistor M3 required to supply constant current  $I$  will be amplified, stabilizing the drain voltage of M1. When the current in the current source, M6, is smaller than the current in the drain of transistor M2, the voltage on the drain of transistor M2 decreases because of the current mismatch. This increases the gate voltage of transistor M4 due to the voltage feedback of transistor M3. Transistor M4 subsequently sources additional current through current mirror transistors M5 and M6 to cancel the current imbalance. When the current imbalance is corrected, the voltage on the gate of transistor M4 returns to its original DC value. The current feedback, in the meantime, keeps the voltage on the drain of transistor M2 more stable which keeps the cascade in its proper operating region. With the cascade functioning throughout the switching interval, the drain of transistor M1 was buffered from the transients and the voltage stored on its gate remained unaffected.

In order to verify the improvement in circuit performance with the current feedback





scheme, simulations were performed for a dynamic current mirror biased with 100uA using a 5kHz sinusoidal 50uA input signal. The circuit was clocked at 100kHz. By eliminating errors due to voltage spiking, the harmonic distortion was improved by almost 9dB over the cascade without current feedback.

## 5 Clock-Feedthrough Error Reduction

Clock-feedthrough has been extensively analyzed in the literature [8,10]. Analysis shows that clock-feedthrough is dependent on the aspect ratio of the switch transistor with respect to the data holding transistor, the clock slope, and the magnitude of the input signal. The signal dependence of clock-feedthrough leads to difficulties in predicting the error, a necessary condition for cancellation. The adaptive clock is a technique for reducing clock-feedthrough through control of the ON conductance of the switch. This control simultaneously reduces the clock swing on the switch and causes the gate-source voltage of the switch to remain constant for varying input signals. With a constant gate-source voltage, the charge injected by the switch becomes constant. This results in the possibility of canceling the error current with a constant current source. In order to use such a system, the nonsaturated region of operation must be used for the data holding transistor. For saturated operation nonlinear transformations between voltage and current result in harmonic distortion even if a constant clock-feedthrough voltage can be generated. For

the nonsaturated region of operation the transformations are linear. A simplified equation for the drain current of a transistor operating in the nonsaturated region is given in Eqn. 3. Added to the gate-source voltage is the constant clock-feedthrough voltage,  $V_{cf}$ .

$$i_{ds} = \beta V_{ds}(V_{gs} + V_{cf} - VT - 1/2V_{ds}). \quad (3)$$

Keeping only the clock-feedthrough term, it can be seen that the error output current is given simply by,

$$i_{cf} = \beta V_{ds} V_{cf}. \quad (4)$$

The clock-feedthrough voltage contributes a DC offset. If constant, no harmonics are generated. A new adaptive clock scheme, applied to the dynamic mirror, is shown in Figure 6. When the clock signal is high, the inverter output is low and the gate-source voltage of switch MC1 is set by the gate-source voltage of transistor M2, regardless of the voltage at the source of MC1. When the clock signal goes low, the inverter turns on which shorts the gate and source of MC1 together. This turns off the transistor and the input signal is held on the gate of M1. By using this control, the gate-source voltage of the switch when on is always equal to the constant gate-source voltage of transistor M2. In order to cancel the constant clock-feedthrough generated by the adaptive clock a constant current that is equivalent to the error current needs to be generated. It has been shown that the integrator circuit presented earlier will perform such a task, [3]. The adaptive clock is also useful as a clock swing limiter. Simulations show that for a data holding transistor (M1 in Figure 6) width to length ratio of 7/5, which gives a transistor area of  $35 \times 10^{-12}$  square meters, the use of an adaptive clock without cancellation reduces total harmonic distortion by 10 dB. The DC offset is reduced by an order of magnitude. As the data holding transistor size is increased, the adaptive clock's effect on total harmonic distortion decreases due to an increase in the data holding capacitance. This limits the influence of clock swing reduction. However, even for larger transistor sizes, 28/20, the use of an adaptive clock without cancellation continues to improve the DC offset and for all device sizes the error is kept constant. The adaptive clock circuit of Figure 6 was fabricated through MOSIS in a two micron CMOS p-well process. The dynamic mirror used the regulated gate current feedback circuit presented earlier as a cascade. Initial experimental results verify the improvements indicated by simulations. For a large data holding transistor size of 28/20 the DC offset error was reduced by 40

## 6 Conclusion

The dynamic current mirror is a useful circuit to reduce reliance on transistor matching. In order to effectively use the dynamic current mirror, consideration has to be given to the effects of transients when switching currents. Clock delays and current feedback were used to reduce distortion due to voltage spikes that occur during intervals of current mismatch. Clock-feedthrough is a source of distortion that effects all methods of sampled-data signal

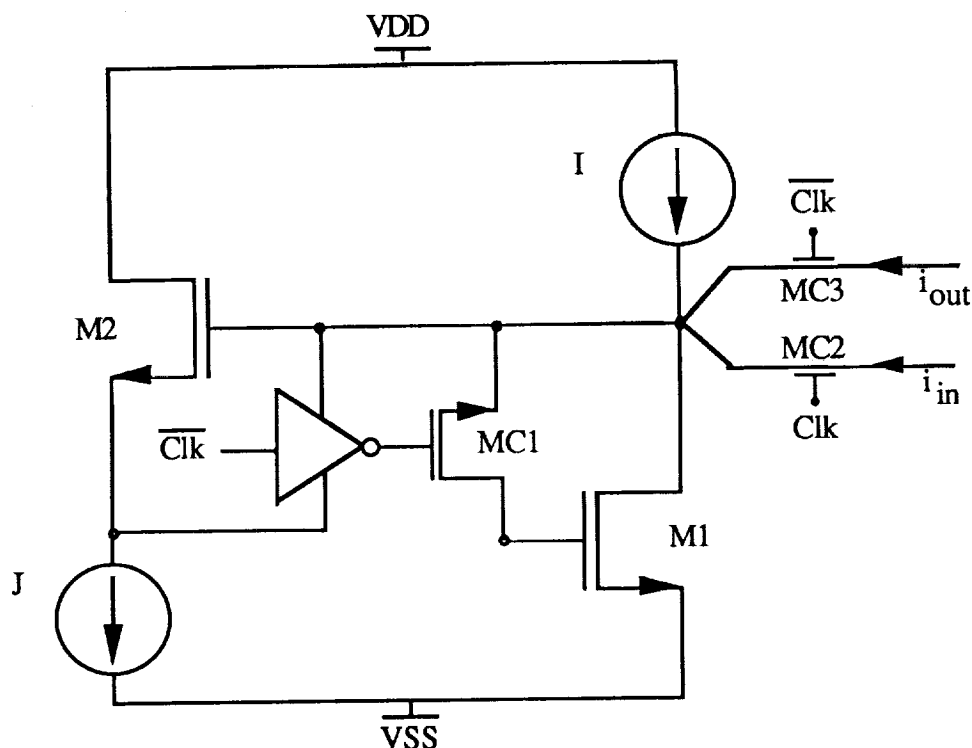


Figure 6: Adaptive clock applied to the dynamic mirror.

processing. For the dynamic mirror an adaptive clock was developed that was shown to both reduce and make constant charge injected by the switch.

## References

- [1] D. Vallancourt, Y.P. Tsividis, and S.J. Daubert, "Current-copier cells," *Electronic Letters*, vol. 24, pp. 1560-1562, Dec. 1988.
- [2] T. S. Fiez and D. J. Allstot, "CMOS switched-current ladder filters," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1360-1367, Dec. 1990.
- [3] J. B. Hughes, I. C. Macbeth, D. M. Pattullo, "Second generation switched-current signal processing," *Proc. of IEEE Intl. Symp. Circuits and Syst.*, May 1990, pp.2805-2808.
- [4] D. G. Nairn and C. A. T. Salama, "A ratio-independent algorithmic analog-to- digital converter combining current-mode and dynamic techniques," *IEEE Trans. Circuits and Syst.*, vol. 37, pp. 319-325, March 1990.
- [5] E. Sachinger and W. Guggenbuhl, "A high-swing, high-impedance MOS cascade circuit," *IEEE J. Solid-State Circuits*, vol. 25, pp. 289-297, Feb. 1990.

- [6] T. S. Fiez, G. Liang, and D. J. Allstot, "Switched-current circuit design issues," IEEE J. Solid-State Circuits, vol. 26, pp. 192-202, March 1991.
- [7] "Analog IC design: the current-mode approach," edited by C. Toumazou, F.J Lidgey and D.G. Haigh, Peter Peregrinus Ltd, 1990.
- [8] C. Eighenberger and W. Guggenbuhl, "On charge injection in analog MOS switches and dummy switch compensation techniques," IEEE Trans. Circuits and Syst., vol. 37, pp. 256-264, Feb. 1990.
- [9] E. A. Vittoz and G. Wegmann, "Analysis and improvements of accurate dynamic current mirrors," IEEE J. Solid-State Circuits, vol. 25, pp. 699-706, June 1990.
- [10] Je-Hurn Shieh, M. Patil, and B. J. Sheu, "Measurement and analysis of charge injection in MOS analog switches," IEEE J. Solid-State Circuits, vol. SC-22, pp. 277-281, April 1987.

# A Tunable CMOS Constant Current Source

D. Thelen

NASA Space Engineering Research Center for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract** - A constant current source has been designed which makes use of on chip electrically erasable memory to adjust the magnitude and temperature coefficient of the output current. The current source includes a voltage reference based on the difference between enhancement and depletion transistor threshold voltages. Accuracy is  $\pm 3\%$  over the full range of power supply, process variations, and temperature using eight bits for tuning.

## 1 Introduction

The lack of precision components in CMOS integrated circuits has traditionally forced design engineers to depend upon external components and matching of on chip components to realize precision functions. For example, switched capacitor filters [1] realize precise transfer functions only when supplied with an accurate clock frequency, which is usually generated by an external crystal oscillator. The locations of poles and zeros are relative to the clock frequency, and are determined by accurate on chip capacitor ratios. Switched current [2], Transconductor C [3], and MOSFET C [4] filters also depend on an external frequency reference, and matching of transistors to realize their transfer functions. In cases where external components are unacceptable, some kind of tuning of the non-ideal components must be accomplished to realize precision functions. Laser trimming is one method which works well, but requires expensive equipment, and a special process. Blowing poly-silicon fuses is inexpensive, but sometimes unreliable, and some types of tuning are difficult to achieve with fuse blowing. Neither laser trimming, nor fuse blowing is reversible, a distinct hindrance if a tuning operation requires more than one iteration. If the circuit to be tuned is fabricated in a process which includes nonvolatile electrically erasable memory, floating gate transistors can be programmed to trim analog performance. Two different methods may be used to employ the floating gate transistor to tune an analog circuit. First, an analog voltage can be stored on the floating gate to change the current or resistance from source to drain [5,6]. The current or resistance will be a function of temperature, and possibly power supply voltage. The second method is to use nonvolatile digital memory to select how much resistance or capacitance is connected to a node, or which tap of a resistor will be connected in a circuit. The second method has the advantage of insensitivity to temperature and power supply voltage, assuming the resistance of the analog switch is low, while it has the disadvantage of requiring more circuitry to do the tuning. In this paper, a circuit is described which uses the later method to tune the magnitude and temperature coefficient of a current source.

## 2 Application

The tunable current source described in this paper, is a part of a larger circuit which emits a constant frequency square wave, independent of temperature, processing, and power supply voltage. The circuit consists of a voltage controlled oscillator (VCO), a frequency to current converter, and an integrator connected in a feedback loop as shown in figure 1.

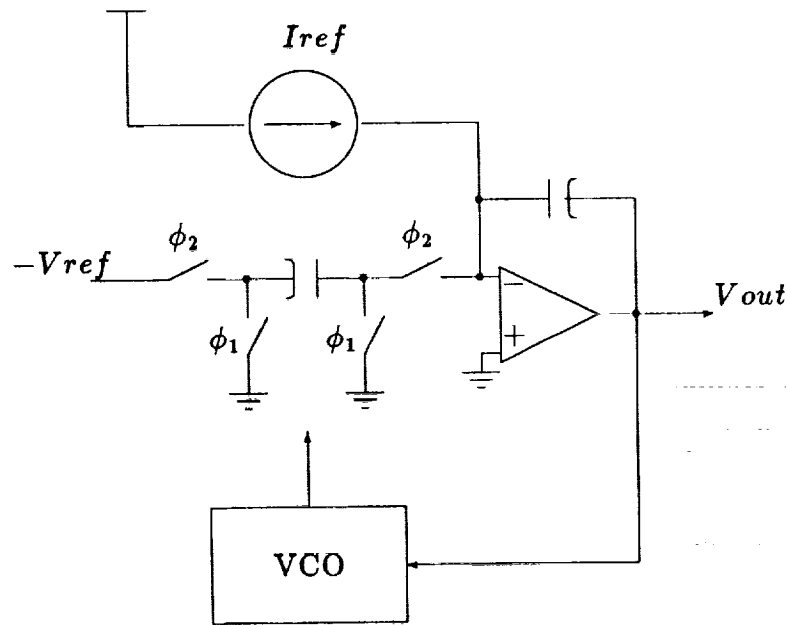


Figure 1: Constant frequency circuit.

The control voltage for the VCO is used in other cells on the chip. The frequency to current converter is based on a switched capacitor network whose average current is given by equation 1 [7]:

$$I_{ave} = f * V * C \quad (1)$$

where  $f$  is frequency,  $V$  is the voltage across the switched capacitor network, and  $C$  is the value of the switched capacitor. At a fixed temperature, both  $V$ , and  $C$  are constants in this circuit, which makes the average current proportional to frequency. The difference between the constant current and the switched capacitor current is integrated, and used to control the VCO. The high gain of the feedback loop ensures that the VCO emits a frequency which causes the current in the switched capacitor network to exactly cancel the constant current. Since the capacitor has a non-zero temperature coefficient, the constant current source must have a temperature coefficient which cancels that of the capacitor. The current source must also compensate for variations in reference voltage and capacitance due to processing.

### 3 Voltage Reference

In this circuit, a constant current will be derived from a constant voltage as shown in figure 2.

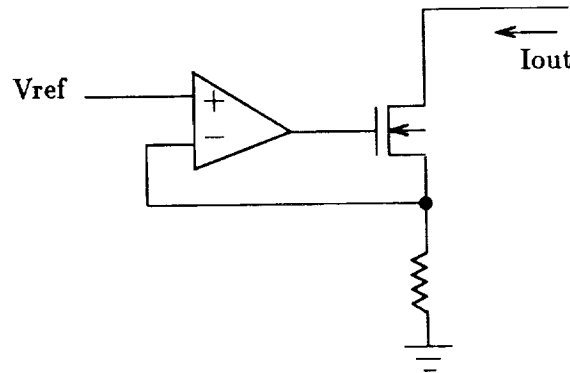


Figure 2: Voltage to current converter.

The performance of the current source will only be as good as the performance of the voltage reference, so the voltage reference must have good rejection of power supply variations and temperature. Three possibilities come to mind to generate a reference voltage on a chip. A power supply voltage divider is the most simple reference available, but since the specification for the current is tighter than the variation of the power supply, the voltage divider can not be used to generate the reference voltage. The bandgap reference [8] is probably the most accurate voltage reference which can be built on a CMOS chip, but it can not be used on this chip because substrate currents, caused by the bipolar transistors, are unacceptable. A threshold voltage reference [9] is based on the difference between the threshold voltages of depletion and an enhancement MOSFET's.

$$V_{ref} \approx V_{te}(1 - \alpha_1 T) - V_{td}(1 - \alpha_2 T) \quad (2)$$

Where  $V_{te}$  is the n-channel enhancement threshold voltage,  $V_{td}$  is the n-channel depletion threshold voltage,  $\alpha_1$  is the temperature coefficient of  $V_{te}$ ,  $\alpha_2$  is the temperature coefficient of  $V_{td}$ , and  $T$  is temperature. Since  $\alpha_1$  and  $\alpha_2$  are approximately equal,  $V_{ref}$  has a very small temperature coefficient.

The mobility temperature coefficient can be ignored by making the width to length ratio of the transistors large for the amount of current flowing through them, and by making the width to length ratio, and drain to source current equal for both transistors. This makes the gate to source voltage mostly  $V_t$ , and the gate to source voltage above  $V_t$  is approximately equal for both transistors. The threshold voltage of both transistors is also dependent on the source to bulk voltage. Depletion and enhancement transistors have approximately the same body effect factor, so if the transistors have the same source to bulk voltage, the body effect will change both threshold voltages equally. This gets canceled by the subtraction as shown in equation 2. One circuit which implements the





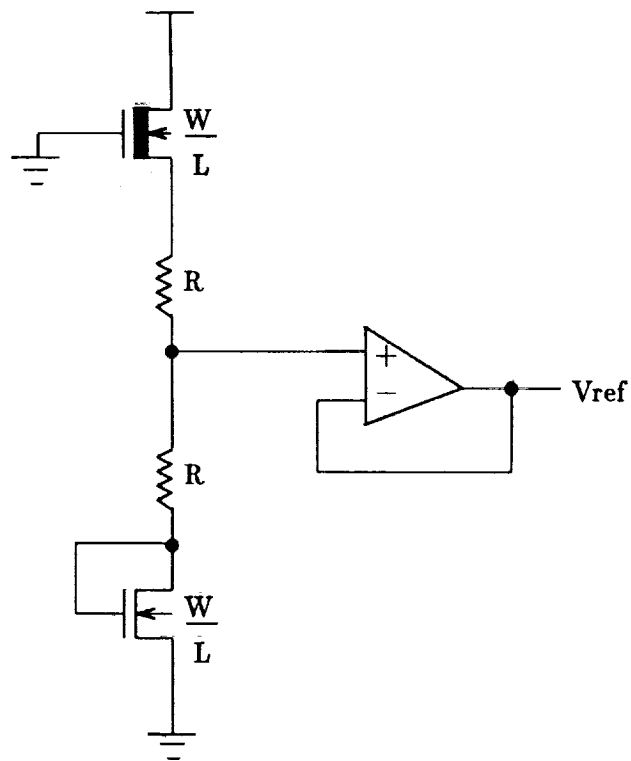


Figure 4: Enhancement-Depletion reference for low power supply voltage.

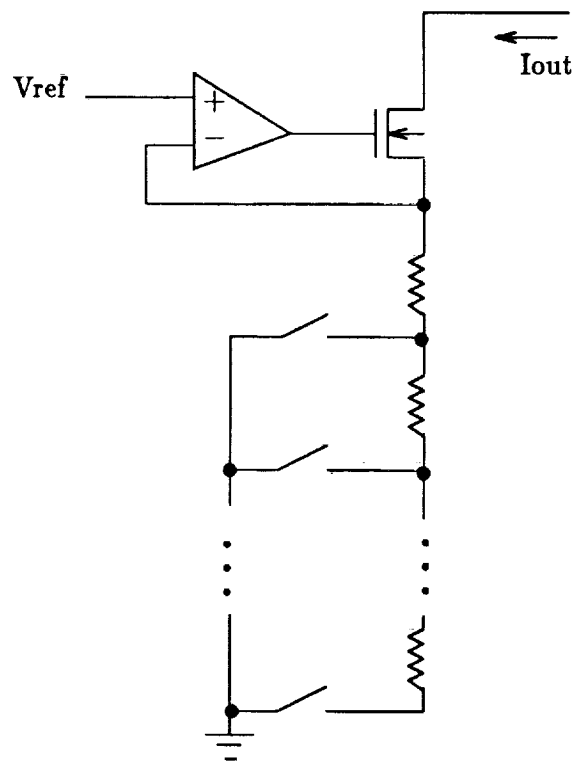


Figure 5: Tuning Scheme for the Magnitude of  $I_{out}$ .

the resistor must have a positive temperature coefficient to cancel the positive temperature coefficient of the resistor. One way to generate a voltage with a positive temperature coefficient is to subtract a voltage with a negative temperature coefficient from a constant voltage. The threshold voltage of an enhancement transistor has a linear negative temperature coefficient suitable for subtraction from  $V_{ref}$ . To make the threshold voltage independent of power supply voltages, a p-channel transistor can be used with its source connected to the bulk to get rid of the body effect. The circuit in figure 6 shows this temperature coefficient cancellation.

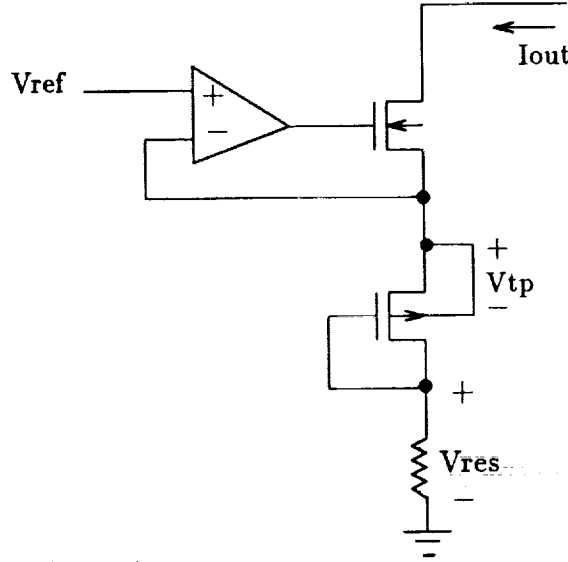


Figure 6: Temperature Coefficient cancellation for  $I_{out}$ .

To tune the temperature coefficient of the current to zero, the magnitude of the reference voltage can be adjusted. This makes the negative temperature coefficient voltage a larger or smaller portion of the reference voltage, which adjusts the overall temperature coefficient. This temperature coefficient cancellation can be expressed as follows:

$$I_{out} \approx \frac{K_1 V_{ref} - V_{tp}(1 - \alpha_1 T)}{K_2 R_{ref}(1 + \alpha_2 T)} \quad (3)$$

where  $V_{tp}$  is the threshold voltage of a p-channel enhancement transistor,  $\alpha_1$  is the temperature coefficient of  $V_{tp}$ , and  $\alpha_2$  is the temperature coefficient of  $R_{ref}$ .  $K_1$  is the fraction of the reference voltage chosen by the first tapped resistor,  $K_2$  is the portion of  $R_{ref}$  chosen by the second tapped resistor.  $K_1$  and  $K_2$  range from zero to one. If we define two new terms:

$$V_{tune} = K_1 V_{ref} - V_{tp} \quad (4)$$

and

$$\beta = V_{tp} \alpha_1 / V_{tune} \quad (5)$$

then equation 3 can be rewritten as:

$$I_{out} \approx \frac{V_{tune} (1 + \beta T)}{K_2 R_{ref} (1 + \alpha_2 T)} \quad (6)$$

When  $\beta$  equals  $\alpha_2$ ,  $I_{out}$  has a zero temperature coefficient.

The key to all this temperature coefficient cancellation is that all the components have only first order temperature coefficients. Measurements from silicon indicate that poly-silicon resistors have linear temperature coefficients, as well as the smallest temperature coefficient of any resistor available on chip. The poly-silicon resistor also has no voltage coefficient, since there is no reverse bias junction which could change the dimension of the resistor. The threshold voltages of n-channel and p-channel transistors have negligible higher order temperature coefficient terms. Threshold voltages of n-channel enhancement and depletion transistors were found to track well over temperature. A simplified schematic of the entire circuit is presented in figure 7.

## 5 Tuning Strategy

Equation 3 shows that  $K_1$  adjusts the magnitude of the current as well as the temperature coefficient, so  $K_1$  must be adjusted first, then  $K_2$  can calibrate the current to the desired value. Tuning the temperature coefficient of this circuit in a production test environment without non-volatile memory would be a logistic nightmare. Somehow the result of the first temperature measurement would have to be stored with a serial number for each die for use during the second temperature test. No such serial numbers are available for each die, and moreover, testing at two temperatures is usually done before and after packaging; one temperature during wafer sort, and the other temperature during final assembly test. Fortunately, with non-volatile memory the results of the first temperature measurement can be written to memory then simply read during the second temperature test. After the temperature coefficient is tuned, the magnitude can be adjusted in one step, since the magnitude adjust ( $K_2$ ) should have almost no effect on the overall temperature coefficient.

## 6 Results

The circuit was simulated with various extremes of power supply voltage, processing, mismatch of threshold voltage temperature coefficients, and temperature. HSpice [10] simulations show the error to be less than 3%. Error can be attributed to non-zero step size in tuning, and finite power supply rejection, especially to the substrate power supply. The chip is presently in layout.

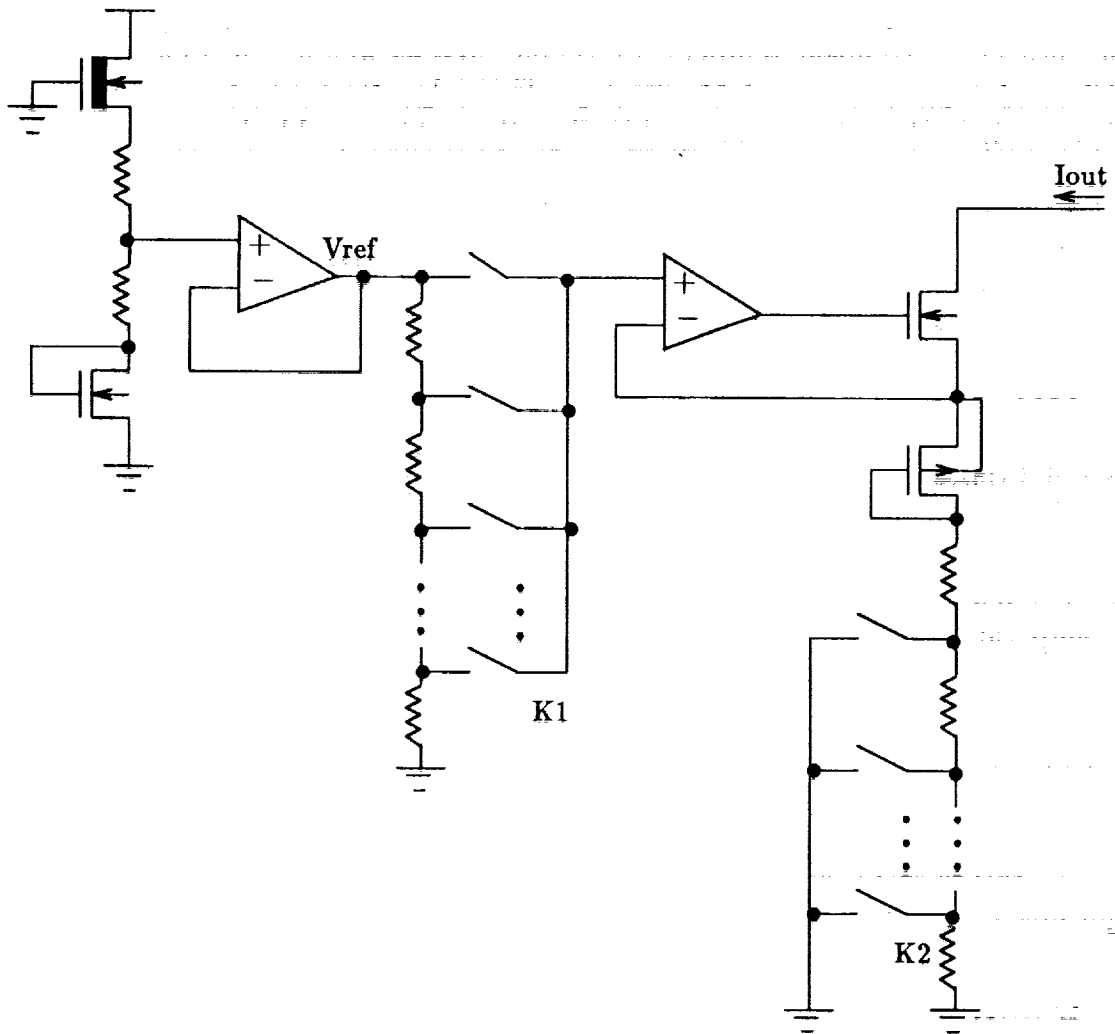


Figure 7: Simplified Schematic.

## 7 Conclusion

Tuning analog circuits with nonvolatile memory provides a very powerful and linear way to overcome the wide tolerances intrinsic in semiconductor processing. The disadvantage of using digital memory to tune analog circuits as opposed to using the floating gate transistors in an analog fashion is the increased number of transistors necessary to do the tuning. The advantage is that the nonlinear characteristics of programming the floating gate transistors can be ignored.

## References

- [1] Bang-Sup Song, "A 10.7-MHz Switched-Capacitor Bandpass Filter," IEEE J. of Solid-State Circuits, Vol. SC-24, pp. 320-324, April, 1989.
- [2] T. Fiez, G. Liang, D. Allstot, "Switched-Current Circuit Design Issues," IEEE J. of Solid-State Circuits, Vol. SC-26, pp. 192-202, March, 1991.
- [3] V. Gopinathan, Y. Tsividis, K. Tan, R. Hester, "Design Considerations for High-Frequency Continuous-Time Filters and Implementation of an Antialiasing Filter for Digital Video," IEEE J. of Solid-State Circuits, Vol. SC-25, pp. 1368-1378, Dec., 1990.
- [4] Jaap van der Plas, "MOSFET-C Filter with Low Excess Noise and Accurate Automatic Tuning," IEEE J. of Solid-State Circuits, Vol. SC-26, pp. 922-929, July, 1991.
- [5] L. R. Carley, "Trimming Analog Circuits using Floating-Gate Analog MOS Memory," IEEE J. of Solid-State Circuits, Vol. SC-24, pp. 1569-1575, Dec., 1989.
- [6] D. Watula, J. Meador, "Auto-Programmable Impulse Neural Circuits," 2nd NASA SERC Symposium on VLSI Design, Moscow, Idaho, pp. 6.3.1-6.3.12, Nov. 1990.
- [7] R. Gregorian, G. Temes, *Analog MOS Integrated Circuits for Signal Processing*. New York: Wiley, 1986.
- [8] M. Ferro, F. Salerno, R. Castello, "A Floating CMOS Bandgap Voltage Reference for Differential Applications," IEEE J. of Solid-State Circuits, Vol. SC-24, pp. 690-697, June, 1989.
- [9] R. Blauschild, P. Tucci, R. Muller, R. Meyer, "A New NMOS Temperature-stable Voltage Reference," IEEE J. of Solid-State Circuits, Vol. SC-13, pp. 767-773, Dec., 1978.
- [10] Meta-Software Inc., 1300 White Oaks Road, Campbell, CA 95008 *HSPICE User's Manual*.

1. The first part of the document discusses the importance of maintaining accurate records of all transactions and activities. It emphasizes the need for transparency and accountability in financial reporting.

2. The second part of the document outlines the various methods and techniques used to collect and analyze data. It includes a detailed description of the experimental procedures and the statistical analysis performed.

3. The third part of the document presents the results of the study. It includes a series of tables and graphs that illustrate the findings of the research. The data shows a clear trend of increasing activity over time.

4. The fourth part of the document discusses the implications of the findings. It suggests that the results have significant implications for the field of study and may lead to further research in this area.

5. The fifth part of the document provides a conclusion and summarizes the key points of the study. It reiterates the importance of accurate record-keeping and the need for ongoing research in this field.

6. The sixth part of the document includes a list of references and a bibliography. It cites various sources that have been consulted during the research process.

7. The seventh part of the document contains a list of appendices and supplementary materials. These include additional data, charts, and documents that provide further detail on the study.

8. The eighth part of the document includes a list of figures and tables. These are numbered and labeled to correspond with the text and provide a visual representation of the data.

9. The ninth part of the document contains a list of footnotes and endnotes. These provide additional information and clarification on specific points mentioned in the text.

10. The tenth part of the document includes a list of acknowledgments and a thank you note. It expresses gratitude to the individuals and organizations that have supported the research.

# DC and Small-Signal Physical Models for the AlGaAs/GaAs High Electron Mobility Transistor

J. C. Sarker and J. E. Purviance

NASA Space Engineering Research Center for VLSI System Design

Department of Electrical Engineering

University of Idaho, Moscow, ID 83843

**Abstract-** Analytical and numerical models are developed for the microwave small-signal performance, such as transconductance, gate-to-source capacitance, current gain cut-off frequency and the optimum cut-off frequency of the AlGaAs/GaAs High Electron Mobility Transistor (HEMT), in both normal and compressed transconductance regions. The validated I-V characteristics and the small-signal performances of four HEMTs are presented.

## Nomenclature

$L$  : Gate length.

$Z$  : Gate width.

$\mu_1$  : Low field mobility of AlGaAs layer.

$\mu_2$  : Low field mobility of two-dimensional electron gas.

$d$  : Thickness of AlGaAs layer.

$d_i$  : Thickness of undoped AlGaAs layer.

$w$  : Width of undepleted region in AlGaAs layer.

$N_d$  : Doping concentration of AlGaAs layer.

$n_s$  : Sheet concentration of two-dimensional electron gas.

$n_{s0}$  : Equilibrium Sheet concentration of two-dimensional electron gas.

$\epsilon_2$  : Permittivity of AlGaAs.

$E_{c1}$  : Saturation electric field of AlGaAs.

$E_{c2}$  : Saturation electric field of two-dimensional electron gas.

$v_s$  : Saturation velocity of two-dimensional electron gas.

$\beta$  : Charge control coefficient.

$\delta$  : Effective width of conduction channel.

$V_{tho}$  : Threshold voltage for two-dimensional electron gas.

$V_{bi}$  : Built-in voltage of Schottky gate on AlGaAs layer.

$V_p$  : Effective pinch-off voltage of AlGaAs layer.

## 1 Introduction

High frequency solid state technology has been moving towards the use of the high electron mobility transistors in microwave and in high speed digital circuits because of its high frequency operation and of its tolerance to many forms of radiation. Several workers have been studying the GaAs HEMTs both theoretically and experimentally since its first introduction in 1980 [1]. Over the past years, analytical, numerical and/or computer-aided

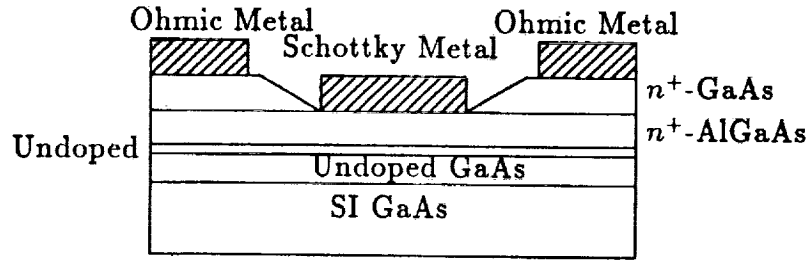


Figure 1: Schematic Diagram of a Uniformly Doped AlGaAs/GaAs HEMT.

models have been reported by many authors. But, because of the complexity in structure of this device, VLSI circuit designers demand a more accurate and compact model for their design.

Among other workers, C.Z. Cil and S. Tansal [2], in 1985, proposed an analytical model which used the simple Trofimenkoff-type velocity-field linear relation [3]. Their modeled results agree very well with the experimental data. However, their model is good only for the linear normal transconductance region; it does not cover the current saturation region and also the parasitic conduction in the AlGaAs layer. But the computer-aided design and simulation of the HEMT circuits demand a complete and more accurate model. In 1986, G.W. Wang and W.H. Ku [4] developed a compact but complete analytical model which covers the whole operation range of the dc characteristics. This model calculates the I-V characteristics of four different HEMTs and compares the modeled results with the experimental data. We have chosen their model as the basis for this work and from this model we have developed analytical and numerical models to calculate the small-signal performances, such as transconductance,  $g_m$ , gate-to-source capacitance,  $C_{gs}$ , current gain cut-off frequency,  $f_T$ , and the optimum value of the cut-off frequency,  $f_{T(opt)}$  before current saturation occurs.

## 2 DC Model

The basic structure of a HEMT device is significantly different from a conventional field effect transistor. A cross sectional view of a uniformly doped AlGaAs/GaAs HEMT device is shown in Figure 1.

At low gate voltage, it has only one current conduction channel but at high gate voltage, it has two conduction channels: one is the two-dimensional electron gas (2-DEG) in the interface between AlGaAs and GaAs and the other is the parasitic conduction through the undepleted  $n^+$ -AlGaAs layer. If the AlGaAs layer is not fully depleted by the Schottky gate and the heterojunction, then the free carriers under the gate are the two-dimensional electrons and the free electrons in the AlGaAs layer. The width of the undepleted AlGaAs



region can be approximated by [5]

$$w \approx d - d_i - \frac{n_{so}}{N_d} - \sqrt{\frac{2\epsilon_2}{qN_d}(V_{bi} - V_G)} \quad (1)$$

By setting  $w = 0$ , the AlGaAs layer is completely depleted, one can obtain the critical value of the gate voltage,  $V_G$  as

$$V_c = V_G(w = 0) = V_{bi} - \frac{qN_d}{2\epsilon_2}\left(d - d_i - \frac{n_{so}}{N_d}\right)^2 \quad (2)$$

The  $V_G \leq V_c$  defines the *normal transconductance region* where only the 2-DEG is the current conduction channel and the  $V_G > V_c$  defines the *compressed transconductance region* where both the 2-DEG and the undepleted AlGaAs layer are the current conduction channels.

According to the charge control model [6], the sheet charge density of the 2-DEG can be approximated as a linear function of gate voltage and channel voltage which is given by

$$n_s(x) = \beta(V_G - V(x) - V_{th}) \quad (3)$$

where  $x$  is in the direction along the heterojunction.

In this dc model, for mathematical simplicity, the Trofimenkoff-type [3] electron velocity-field relation has been used for both the 2-DEG channel and the AlGaAs parasitic conduction channel. The linear electron velocity-field can be related as

$$v(x) = \frac{\mu E(x)}{1 + \frac{E(x)}{E_c}} \quad (4)$$

Here,  $E(x)$  is the electric field in the 2-DEG channel or in the undepleted AlGaAs layer and  $E_c$  is the field at which the velocity of electrons reach the maximum value (saturation velocity).

Using the charge control concept and the velocity-field relationship described above, the current conducting through the 2-DEG channel can be determined by

$$I_{2-DEG} = Zqn_s(x)v(x) \quad (5)$$

Similarly, the current through the undepleted AlGaAs layer can be determined by

$$I_{AlGaAs} = ZqN_dw(x)v(x) \quad (6)$$

Here, for simplicity, full ionization of the donor atoms has been assumed for the current through the AlGaAs layer.

#### (A) I-V Equations in the Normal Transconductance Region

When the gate voltage is low, i.e.  $V_G \leq V_c$ , the normal transconductance region is formed. This region is then divided into the *linear* ( $V_D < V_{sat}$ ) and the *saturation* ( $V_D \geq V_{sat}$ ) regions. The current-voltage relationship in two different regions can be derived as follows:

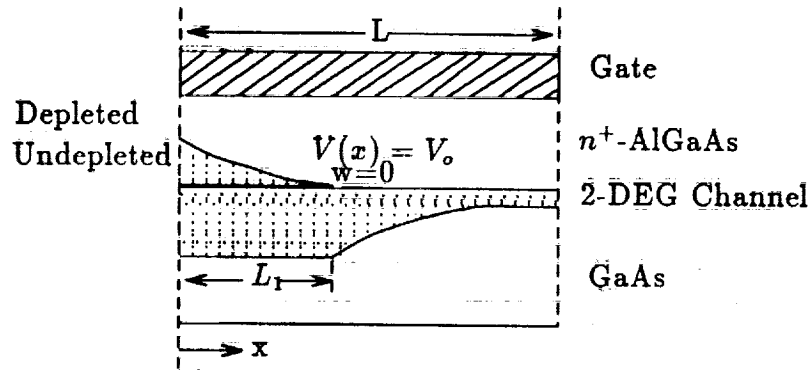


Figure 2: Schematic Diagram Showing Current Saturation in the 2-DEG Channel.

(i) Normal Linear Region ( $V_D < V_{sat}$ )

Introducing equations (3) and (4) in equation (5), and integrating from source to drain along the 2-DEG channel, the current through the channel is

$$I_D = \frac{A(V_G - V_{th} - \frac{V_D}{2})V_D}{1 + \frac{V_D}{B}} \quad (7)$$

where  $A = \frac{Zq\beta\mu_2}{L}$  and  $B = LE_{c2}$  are the model parameters;  $V_G$  and  $V_D$  are the internal gate and drain voltages.

(ii) Normal Saturation Region ( $V_D \geq V_{sat}$ )

The velocity-field relation (equation (4)) allows the velocity to saturate when the electric field approaches infinity. But physically it is impossible; so, the model assumed that the velocity saturation occurs when  $E \geq E_c$ . Therefore, from equation (4), the saturation velocity at  $E = E_c$  is  $v_s = \frac{\mu E_c}{2}$ .

When the drain voltage,  $V_D$  becomes greater than the saturation voltage,  $V_{sat}$  the situation becomes like Figure 2. At  $x = L_c$ , electric field exceeds saturation field,  $E_c$ , and the electron velocity saturates; and after this the electrons move with this constant saturation velocity. Then, using  $V = V_{sat}$  and  $\frac{dV}{dx} = E_{c2}$  at  $x = L_c$ , equation (7) can be written as

$$I_D = \frac{Zq\beta\mu_2(V_G - V_{th} - \frac{V_{sat}}{2})V_{sat}}{L_c + \frac{V_{sat}}{E_{c2}}} \quad (8)$$

Also from equations (3) and (5), the current in the saturation region can be written as

$$I_D = Zq\beta(V_G - V_{th} - V_{sat})v_s = \frac{Zq\beta\mu_2}{2}(V_G - V_{th} - V_{sat})E_{c2} \quad (9)$$

Now, using the current continuity condition, equations (8) and (9) can be combined to obtain

$$V_{sat} = \frac{(1 - K_1)B(V_G - V_{th})}{(1 - K_1)B + (V_G - V_{th})} \quad (10)$$

where  $K_1 = \frac{L-L_c}{L}$ . Generally,  $L_c$  and  $V_{sat}$  can be determined by solving a two-dimensional Poisson's equation which has the form in the velocity saturation region

$$\frac{\partial^2 V}{\partial x^2} \approx \alpha I_D \quad (11)$$

where  $\alpha = \frac{1}{\epsilon_2 v_s \delta}$ . Here,  $\delta$  is the effective width of the conduction channel which is assumed to be invariant to the bias voltage as compared to  $I_D$  and set to a constant. This Poisson's equation is obtained by neglecting the variation of carrier concentration in the direction perpendicular to the channel and can be solved with boundary conditions  $V(L = L_c) = V_{sat}$  and  $E(L = L_c) = E_c$ . The final form of the solution becomes

$$V_D - V_{sat} = \frac{\alpha I_D (L - L_c)^2}{2} + E_{c2}(L - L_c) = C I_D K_1^2 + B K_1 \quad (12)$$

where  $C = \frac{\alpha L^2}{2} = \frac{L^2}{2\epsilon_2 v_s \delta}$  is the third model parameter. Equations (10) and (12) can be solved simultaneously to find  $K_1$  and  $V_{sat}$  :

$$K_1 = \frac{-X + \sqrt{X^2 + [2CA(V_G - V_{th})^2 - 4B][1 + \frac{(V_G - V_{th})}{B}]V_D}}{CA(V_G - V_{th})^2 - 2B} \quad (13)$$

where  $X = B + V_D + V_G - V_{th}$ . Then from equation (8), the saturation current equation can be written as

$$I_D = \frac{A(V_G - V_{th} - \frac{V_{sat}}{2})V_{sat}}{1 - K_1 + \frac{V_{sat}}{B}} \quad (14)$$

### (B) I-V Equations in the Compressed Transconductance Region

When the gate voltage is high enough such that  $V_G > V_c$ , the AlGaAs layer starts to conduct current. This current conduction mechanism can be considered similar to a parasitic MESFET and it is shown in Figure 3.

When  $w = 0$  at  $x = L_1$ , from equation (1) the voltage inside the channel is  $V = V_o = V_p - V_{bi} + V_G$ , where  $V_p$  is defined as

$$V_p = \frac{qN_d}{2\epsilon_2} \left( d - d_i - \frac{n_{so}}{N_d} \right)^2 \quad (15)$$

When  $V_D \leq V_o$ , the sheet carrier concentration,  $n_s$ , of the whole 2-DEG channel is equal to its equilibrium value,  $n_{so}$  and it is independent of gate and drain voltage. The 2-DEG channel is then like a non-linear resistor with sheet concentration,  $n_{so}$ , while the undepleted AlGaAs behaves like a MESFET. This equilibrium concentration is assumed to be maximum and is given by (from equation (3))

$$n_{so} = \beta(V_{bi} - V_p - V_{th}) \quad (16)$$

From the schematic diagram shown in Figure 3, the compressed transconductance region can be divided into three different regions of operation :

(i) Linear Region I :  $V_D \leq V_o$

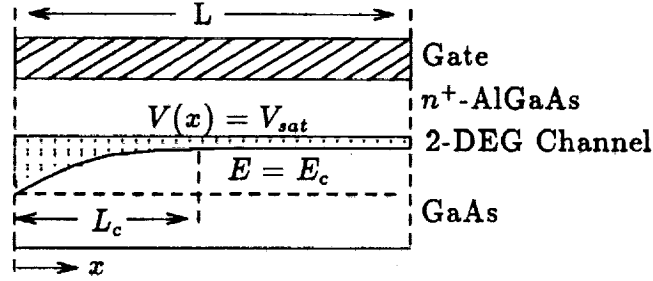


Figure 3: Schematic Diagram Showing Current Saturation in the 2-DEG Channel and the Parasitic Conduction through the AlGaAs Layer.

(ii) Linear Region II :  $V_o < V_D < V_{sat}$

(iii) Saturation Region :  $V_D \geq V_{sat}$

Here, the assumption  $V_{sat} \geq V_o$  has been made to allow division into various regions of operation. This assumption is true for typical HEMT devices.

(i) Linear Region I

For  $V_D \leq V_o$ , the current through the AlGaAs layer is derived as in the case of the MESFET and is given by

$$I_1 = \frac{E}{1 + \frac{V_D}{F}} \left[ V_D - \frac{2(V_{bi} - V_G + V_D)^{3/2} - (V_{bi} - V_G)^{3/2}}{\sqrt{V_p}} \right] \quad (17)$$

where  $E = \frac{Zq\mu_1 N_d}{L}(d - d_i - \frac{n_{i0}}{N_d})$  and  $F = LE_{c1}$  are two more model parameters.

The current through the 2-DEG channel becomes

$$I_2 = \frac{A(V_{bi} - V_p - V_{th})V_D}{1 + \frac{V_D}{B}} \quad (18)$$

The total current in this region of operation is the sum of these two currents :  $I_D = I_1 + I_2$ .

(ii) Linear Region II

From Figure 3, for  $V_D > V_o$ , the current flowing through the 2-DEG channel is

$$I_2 = \frac{Zq\mu_2 n_{so} V_o}{L_1 + \frac{V_o}{E_{c2}}} \quad (19)$$

From this equation

$$L_1 = \frac{Zq\mu_2 n_{so} V_o}{I_2} - \frac{V_o}{E_{c2}} \quad (20)$$

Current through the AlGaAs layer can be obtained from equation (6) as

$$I_1 = \frac{Zq\mu_1 N_d w \frac{dV}{dx}}{1 + \frac{1}{E_{c1}} \frac{dV}{dx}}$$

Integrating this equation for  $V$  from 0 to  $V_o$  and for  $x$  from 0 to  $L_1$ , and then using equation (20) for  $L_1$ , the final expression for current becomes

$$I_1 = \frac{E \left[ V_o - \frac{2}{3} \left( V_p - \frac{(V_{bi} - V_G)^{3/2}}{\sqrt{V_p}} \right) \right]}{\left[ \frac{A(V_{bi} - V_p - V_{th})}{I_2} - \frac{1}{B} + \frac{1}{F} \right] V_o} \quad (21)$$

The derivation of the current expression in the 2-DEG is similar to the normal region. But here, the limits of integration for  $V$  are from  $V_o$  to  $V_D$  and for  $x$  from  $L_1$  to  $L$ . After performing the integration and using equation (20) for  $L_1$ , the current through the 2-DEG channel can be obtained as

$$I_2 = \frac{A[(V_G - V_{th} - \frac{V_D + V_o}{2})(V_D - V_o) + (V_{bi} - V_p - V_{th})V_o]}{1 + \frac{V_D}{B}} \quad (22)$$

So, the total drain current is the sum of equations (21) and (22).

### (iii) Saturation Region

For the saturation region,  $V_D \geq V_{sat}$ , the current expression for the undepleted AlGaAs layer is the same as the linear region II (equation (21)). The principle to find the saturation voltage in this operating region is similar to that in the normal region except the contribution from the parasitic conduction has to be taken into account. From the current continuity at the interface of the velocity saturation region and the non-saturation region (Figure 3), the saturation voltage can be obtained as

$$V_{sat} = \frac{[(1 - K_1)B(V_G - V_{th}) + 2(V_G - V_{bi} + V_p)V_o - V_o^2]}{[(V_G - V_{th}) + (1 - K_1)B]} \quad (23)$$

On the other hand, the solution of the Poisson's equation (equation (12)) in this region becomes

$$V_D - V_{sat} = \frac{CAK_1^2 \left[ (V_G - V_{th})V_{sat} - \frac{V_{sat}^2}{2} - (V_G - V_{bi} + V_p)V_o + \frac{V_o^2}{2} \right]}{1 - K_1 + \frac{V_{sat}}{B}} + BK_1 \quad (24)$$

By solving equations (23) and (24) iteratively, values of  $K_1$  and  $V_{sat}$  can be found. Once  $K_1$  and  $V_{sat}$  are found, the current through the 2-DEG channel can be obtained as

$$I_2 = \frac{A[(V_G - V_{th} - \frac{V_{sat} + V_o}{2})(V_{sat} - V_o) + (V_{bi} - V_p - V_{th})V_o]}{1 - K_1 + \frac{V_{sat}}{B}} \quad (25)$$

The total drain current is then the sum of equations (21) and (25).

In the subthreshold region of operation the charge control is not linear; so, in addition to the model and physical parameters, a fitting parameter,  $D$  is used to model the threshold voltage shift of the 2-DEG caused by the drain voltage. This simple threshold voltage correction is given by

$$V_{th} = V_{tho} - D \times V_D \quad (26)$$

So, with the nine parameters  $A$ ,  $B$ ,  $C$ ,  $E$ ,  $F$ ,  $V_p$ ,  $V_{tho}$ ,  $V_{bi}$  and  $D$ , the I-V characteristics of the AlGaAs/GaAs HEMT device can be modeled completely.

### 3 Small-Signal Model

Evaluation and analysis of the small-signal performances of the HEMT are important for the operation of microwave circuits. The HEMT is usually biased in the normal transconductance region without parasitic conduction for optimal low-noise and/or high-frequency performance. Some of the small-signal parameters like transconductance, gate-to-source capacitance, current gain cut-off frequency etc. can be derived analytically from this model. The derivation of these parameters in the saturated normal region and also in the compressed transconductance region is mathematically complicated and computationally involves more CPU time. So, to determine these parameters in those regions of operation a computationally efficient numerical technique has been used. Methods of determining of these small-signal parameters are discussed in the next few subsections.

#### 3.1 Transconductance, $g_m$

The intrinsic transconductance,  $g_m$  at constant drain voltage is defined as

$$g_m \equiv \frac{\partial I_D}{\partial V_G} \bigg|_{V_D = \text{Constant}}$$

The  $g_m$  in the linear normal region can be obtained analytically by differentiating drain current (equation (7)) with respect to gate voltage :

$$g_m = \frac{\partial}{\partial V_G} \left[ \frac{A(V_G - V_{th} - \frac{V_D}{2})V_D}{1 + \frac{V_D}{B}} \right] = \frac{AV_D}{1 + \frac{V_D}{B}} \quad (27)$$

The transconductance increases with drain voltage before current saturation and is inversely proportional to gate length and mobility degradation factor  $(1 + \frac{V_D}{B})$ .

To calculate  $g_m$  in the saturation region and in the compressed (both linear and saturation) region, we differentiate the corresponding drain currents numerically. For this numerical differentiation we have used the centered-finite-divided difference equation of the form [7]

$$g_m(V_{G_i}) = \frac{I_D(V_{G_{i+1}}) - I_D(V_{G_{i-1}})}{V_{G_{i+1}} - V_{G_{i-1}}} \quad (28)$$

Here,  $g_m(V_{G_i})$  is the transconductance evaluated at the  $i^{th}$  point.

#### 3.2 Gate-to-Source Capacitance, $C_{gs}$

Gate-to-source capacitance,  $C_{gs}$ , is defined, with the assumption  $C_{gd} \ll C_{gs}$ , as

$$C_{gs} \equiv \frac{\partial Q_T}{\partial V_G}$$

where  $Q_T$  is the total charge.

In the normal region, the AlGaAs layer is completely depleted, so the  $C_{gs}$  is due only to the two-dimensional electron gas. Thus, for the normal region

$$C_{gs} = \frac{\partial}{\partial V_G} \left[ \int_0^L qn_s(x)dx \right]$$

Substituting equation (3) for  $n_s(x)$  and then performing the integration, we get

$$C_{gs} = \frac{\partial}{\partial V_G} \left[ \int_0^L q\beta(V_G - V(x) - V_{th})dx \right] = \frac{AL^2(2 + \frac{V_D}{B})}{\mu_2} \quad (29)$$

Calculation of gate-to-source capacitance in the saturation region is more complicated because of complexity in the total charge calculation in the channel. The method we have used to calculate the charge in the channel is given in detail in reference [8]. The final expression of total charge,  $Q_T$  becomes

$$Q_T = \frac{AL}{\mu_2} \left[ (V_G - V_{th})L - V_{sat}(L - \frac{L_c}{2}) \right] \quad (30)$$

where

$$L_c = -\frac{V_{sat}L}{B} + \frac{AL(V_G - V_{th} - \frac{V_{sat}}{2})V_{sat}}{I_D} \quad (31)$$

In this equation, the saturation current,  $I_D$  is calculated by using equation (14) at the saturation voltage,  $V_{sat}$ .

Once we know the total charge in the channel we can calculate the  $C_{gs}$  by using numerical differentiation. The form of this differentiation is analogous to the  $g_m$  equation

$$C_{gs}(V_{G_i}) = \frac{Q_T(V_{G_{i+1}}) - Q_T(V_{G_{i-1}})}{V_{G_{i+1}} - V_{G_{i-1}}} \quad (32)$$

Ideally, to calculate  $C_{gs}$  in the compressed region, the capacitance due to the charge accumulated in the undepleted AlGaAs layer has to be added with the capacitance due to the 2-DEG channel. But the calculation of the capacitance due to AlGaAs layer analytically from this model is not very straightforward. Moreover, this additional capacitance contribution may not be very significant, particularly at high drain voltages. So, in this work we have neglected this contribution compared to the capacitance due to the 2-DEG channel charge. Therefore, equation (32) has also been used to calculate the gate-to-source capacitances in the compressed transconductance region.

### 3.3 Current Gain Cut-off Frequency, $f_T$

In microwave applications, the current gain cut-off frequency is the frequency used as an indicator of the device speed. The conventional definition of  $f_T$  is

$$f_T \equiv \frac{g_m}{2\pi C_{gs}}$$

In the normal linear region, we calculated  $f_T$  analytically by using equations (27) and (29):

$$f_T = \frac{1}{2\pi} \left[ \frac{AV_D}{1 + \frac{V_D}{B}} \right] \left[ \frac{\mu_2}{AL^2(2 + \frac{V_D}{B})} \right] = \frac{\mu_2 V_D}{2\pi L^2(1 + \frac{V_D}{B})(2 + \frac{V_D}{B})} \quad (33)$$

We again adopted the numerical techniques to calculate  $f_T$  for normal saturated region and both linear and saturated compressed regions. This numerical expression is given by

$$f_T(V_{G_i}) = \frac{g_m(V_{G_i})}{2\pi C_{gs}(V_{G_i})} \quad (34)$$

Here, the  $f_T$ ,  $g_m$  and  $C_{gs}$  are calculated at the  $i^{th}$  point.

### 3.4 Optimum Cut-off Frequency, $f_T(\text{opt})$

Another important parameter in microwave applications is the optimum frequency,  $f_T(\text{opt})$ . This optimum frequency is defined as the maximum value of the current gain cut-off frequency just before current saturation occurs. Thus, in the normal transconductance region,  $f_T(\text{opt})$  is approximated as

$$f_T(\text{opt}) = \frac{\mu_2 V_{sat}}{2\pi L^2(1 + \frac{V_{sat}}{B})(2 + \frac{V_{sat}}{B})} \quad (35)$$

Here,  $V_{sat}$ , the value of the saturation voltage when current just starts to saturate, can be evaluated by setting  $K_1 = 0$  in equation (10) :

$$V_{sat} = \frac{B(V_G - V_{th})}{B + (V_G - V_{th})} \quad (36)$$

## 4 Results and Discussion

### 4.1 The I-V Characteristics

To validate the dc model we have developed a computer simulation program which calculates the I-V characteristics over the entire region of operation. Using this simulation program we have calculated the I-V characteristics of all the four HEMTs. The device physical parameters and the modeling parameters of these HEMTs, taken from reference [4], are given in Table 1.

In the derivation of the drain current equations in section 2, the dc model does not include the effects of parasitic source and drain resistances explicitly. These effects can be taken into account in the model by solving the nonlinear equations which are given below

$$V_{GS} = V_G + I_D(V_G, V_D)R_S \quad (37)$$

and

$$V_{DS} = V_D + I_D(V_G, V_D)(R_S + R_D) \quad (38)$$



Device	HEMT #1 (TRW #2078)	HEMT #2	HEMT #3 (GE #5410)	HEMT #4
$L(\mu\text{m})$	0.35	1.0	0.25	1.0
$Z(\mu\text{m})$	65	145	100	1200
$V_{tho}(V)$	-0.017	-0.901	-0.912	-2.389
$V_p(V)$	-	-	1.481	2.319
$V_{bi}(V)$	-	-	0.85	0.85
$A(\text{mA}/V^2)$	49.517	101.253	103.539	454.167
$B(V)$	5.285	1.604	0.616	0.948
$C(K\Omega)$	8.341	0.583	0.992	0.201
$D$	0.015	0	0.092	0.008
$E(\text{mA}/V)$	-	-	81.825	542.663
$F(V)$	-	-	2.154	3.04
$R_S(\Omega)$	5.9	7.0	4.6	1.0
$R_D(\Omega)$	6.0	7.0	6.0	1.0

Table 1: Physical and Model Parameters of the HEMTs.

where  $V_{GS}$  and  $V_{DS}$  are the externally applied gate and drain voltages respectively;  $R_S$  and  $R_D$  are the parasitic source and drain resistances. These two equations were solved iteratively in the program to find the values of  $V_G$  and  $V_D$  for given values of external voltages  $V_{GS}$  and  $V_{DS}$ .

The HEMT #1 and #2 show only normal transconductance effects; only five model parameters,  $V_{tho}$ ,  $A$ ,  $B$ ,  $C$  and  $D$  are needed in the program to calculate the I-V relation. With these parameter values and using equations (7), (10), (13), (14), (37) and (38), we have developed a simulation program which calculates the drain-to-source current as a function of external drain voltage for different external gate voltages.

Figure 4 shows the I-V curve of the HEMT #1. In the program, we have swept the drain voltage from 0 to 3 volts with a 0.2 volts steps and calculated drain-to-source currents for gate voltages  $V_{GS} = 0, 0.1, 0.2, 0.3, 0.4$  and  $0.5$  volts. As a comparison, we have also plotted the experimental data obtained from reference [4]. From the figure, we can see a nice agreement between our I-V results and the experimental data.

Simulated results along with experimental data [4,6] of the HEMT #2 are shown in Figure 5. In this case the drain voltage was varied from 0 to 3 volts with 0.25 volts steps. Drain currents for  $V_{GS} = -0.8, -0.6, -0.4, -0.2$  and  $0$  volts were calculated. The low gate bias curves agree very well with the experimental values. As the gate bias increases a small deviation occurs near the linear and saturation transition region.

The I-V characteristics of the HEMT #3 and #4 (double heterojunction HEMT) are more complex because of the compressed transconductance effect (in addition to the normal transconductance effect). Four additional parameters  $V_p$ ,  $V_{bi}$ ,  $E$  and  $F$  are needed to model this effect. So, with the nine parameter values listed in Table 1 and using the equations (17), (18) and (21-25), we have calculated the drain-to-source currents in the compressed transconductance region. Equation (24) was rearranged such that  $K_1$  can be written in

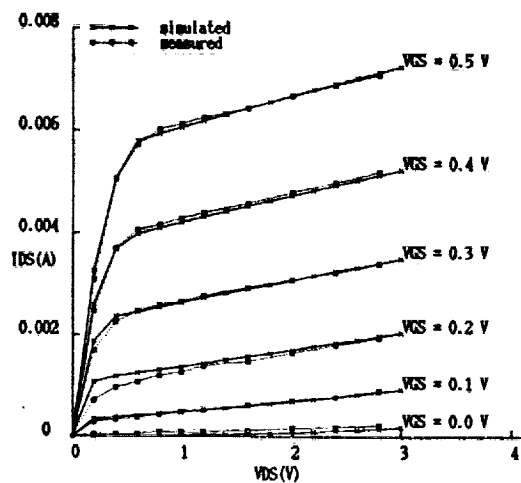


Figure 4: Characteristics of HEMT #1

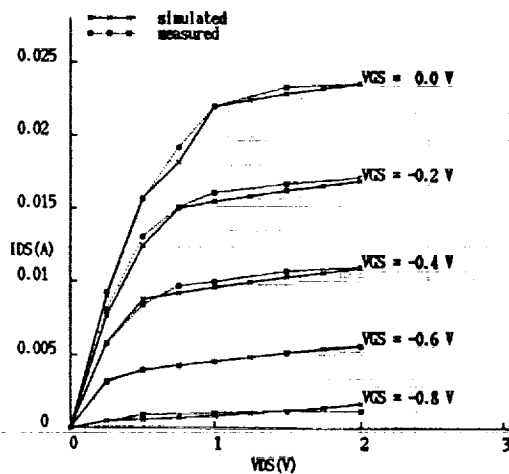


Figure 5: I-V Characteristics of HEMT #2

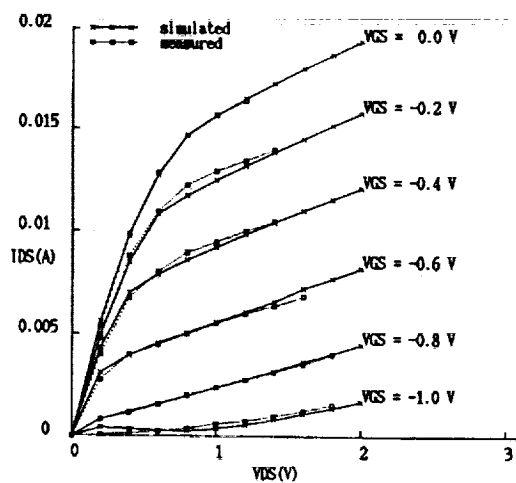


Figure 6: Characteristics of HEMT #3

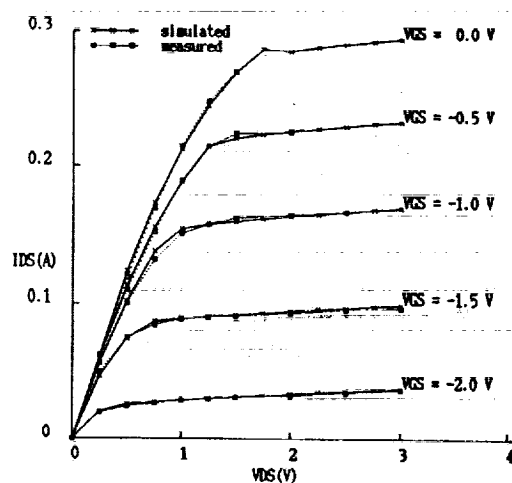


Figure 7: I-V Characteristics of HEMT #4

terms of  $V_{sat}$

$$K_1 = \frac{-(B + V_D) + \sqrt{(B + V_D)^2 + 4[CAY - B][(1 + \frac{V_{sat}}{B})(V_D - V_{sat})]}}{2\{CAY - B\}} \quad (39)$$

where  $Y = (V_G - V_{th})V_{sat} - \frac{V_{sat}^2}{2} - \frac{V_D^2}{2}$ . Equations (23) and (39) were solved iteratively in the program by assuming an initial value of  $K_1 = 0$  to obtain  $V_{sat}$  and then  $K_1$ . After  $K_1$  and  $V_{sat}$  are known, the drain current through the 2-DEG channel in the saturation region is calculated by using equation (25).

Figure 6 shows the I-V curve of the HEMT #3. Here, we have scanned the drain voltage from 0 to 3 volts at a step of 0.2 volts for the gate voltages,  $V_{GS} = -1.0, -0.8, -0.6, -0.4, -0.2$  and 0 volts. As we can see the modeled result agrees very well with the experimental data [4].

Finally, we have calculated the I-V characteristics of a double heterojunction HEMT (HEMT #4) and the results, along with the experimental data are shown in Figure 7. These results also agree fairly well with the published measured data [4].

Two of the four HEMTs (HEMT #1 and #3) are sub-half-micron gate HEMTs. Unmodeled short channel effects such as velocity overshoot and unmodeled hot carrier effects may occur in these two HEMTs. It is reported that these effects start to become prominent below  $0.25\mu m$  gate length [9], therefore HEMT #3 may show considerable short channel effect in the compressed transconductance region. Moreover, this dc model was originally developed only for the single-heterojunction HEMT. But from our simulation results of HEMT #4, which is a double-heterojunction HEMT, we found that this model also appears to be good for the double-heterojunction HEMT.

## 4.2 Small-Signal Performance Calculation

Based on the equations derived in section 3 and the physical parameters listed in Table 1, we have developed the simulation program which calculates the small-signal performances. Using this program we have calculated  $I_D$ ,  $g_m$ ,  $C_{gs}$ ,  $f_T$  and  $f_T(\text{opt})$  as a function of gate voltage keeping drain voltage fixed. Table 2 shows the small-signal parameter values for all the four HEMTs for different drain and gate bias conditions.

This small-signal model has been developed in an academic environment, based on a quasi-static approximation. The values of the small-signal parameters are essentially theoretical and have not been rigorously validated in this work because of the unavailability of the experimental data.

## 5 Conclusion

A complete analytical dc model for the uniformly doped AlGaAs/GaAs HEMT device has extensively analyzed and validated independently. Based on the model a simulation program was developed to calculate the I-V characteristics. Using this program, the I-V

Device	Bias Condition	$I_D(mA)$	$g_m(mS)$	$C_{gs}(fF)$	$f_T(GHz)$	$f_T(opt)(GHz)$
HEMT #1	Normal Linear Region $V_{GS} = 0.4V, V_{DS} = 0.4V$	3.677	16.526	28.501	92.283	96.055
HEMT #1	Normal Saturation Region $V_{GS} = 0.2V, V_{DS} = 1.0V$	1.376	10.868	6.5129	265.58	-
HEMT #2	Normal Linear Region $V_{GS} = -0.2V, V_{DS} = 0.5V$	12.389	27.471	507.09	8.622	10.698
HEMT #2	Normal Saturation Region $V_{GS} = -0.2V, V_{DS} = 1.0V$	15.461	31.098	134.89	36.692	-
HEMT #3	Normal Linear Region $V_{GS} = -0.65V, V_{DS} = 0.2V$	2.517	14.005	33.553	66.432	69.527
HEMT #3	Normal Saturation Region $V_{GS} = -0.8V, V_{DS} = 1.0V$	2.395	14.513	9.441	244.66	-
HEMT #3	Compressed Linear Region $V_{GS} = -0.2V, V_{DS} = 0.4V$	8.476	6.699	27.512	38.750	-
HEMT #3	Compressed Saturation Region $V_{GS} = -0.4V, V_{DS} = 1.0V$	9.215	16.924	11.344	237.45	-
HEMT #4	Normal Linear Region $V_{GS} = -1.5V, V_{DS} = 0.5V$	74.628	116.277	1583.0	11.89	13.18
HEMT #4	Normal Saturation Region $V_{GS} = -1.5V, V_{DS} = 1.0V$	88.939	133.306	500.9	42.36	-
HEMT #4	Compressed Linear Region $V_{GS} = -0.5V, V_{DS} = 1.0V$	188.287	48.913	1113.0	6.996	-
HEMT #4	Compressed Saturation Region $V_{GS} = -1.0V, V_{DS} = 1.0V$	153.567	123.322	548.7	35.77	-

Table 2: Small-Signal Performances of the AlGaAs/GaAs HEMTs Calculated in this Work.

curves for four HEMTs were successfully calculated and compared with the experimental data reported earlier [4,6].

In the second phase of the work, analytical and numerical methods were developed to predict some of the important small-signal performances of these HEMTs. Based on this new computer-aided model, the small-signal parameters,  $g_m$ ,  $C_{gs}$ ,  $f_T$  and  $f_T(opt)$  were calculated and are presented in Table 2. The proposed small-signal model for the AlGaAs/GaAs HEMT device may be useful to VLSI and microwave applications in future.

## 6 Acknowledgment

The authors acknowledge NASA for funding this project under the grant NAG5-1043. They would like to thank the NASA Space Engineering Research Center, University of Idaho for partial funding and for providing facilities to accomplish this work. The second author acknowledges Jesus of Nazareth for His sacrifice and example.

## References

- [1] T. Mimura, S. Hiyamiza, T. Fujii, and K. Nambu, "A New Field Effect Transistor with Selectively Doped GaAs/n- $Al_xGa_{1-x}$ As Heterojunctions," *Jpn. Appl. Phys.*, Vol. 19, 1980, pp. L225-L227.
- [2] C.Z. Cil and S. Tansal, "A New Model for Modulation-Doped FET's," *IEEE Electron Device Letters*, Vol. EDL-6, Aug. 1985, pp. 434-436.
- [3] F.N. Trofimenkoff, "Field-Dependent Mobility Analysis of the Field-Effect Transistor," *Proc. IEEE*, Vol. 53, Nov. 1965, pp. 1765-1766.
- [4] G.W. Wang and W.H. Ku, "An Analytical and Computer-Aided Model of the Al-GaAs/GaAs High Electron Mobility Transistor," *IEEE Transactions on Electron Devices*, Vol. ED-33, May 1986, pp. 657-663.
- [5] D.L. Pulfrey and N.G. Tarr, *Introduction to Microelectronic Devices*, Prentice-Hall, Inc. 1989.
- [6] K. Lee, M.S. Shur, T.J. Drummond, and H. Morkoc, "Current-Voltage and Capacitance-Voltage Characteristics of Modulation-Doped Field-Effect Transistors," *IEEE Transactions on Electron Devices*, Vol. ED-30, March 1983, pp. 207-212.
- [7] S.C. Chapra and R.P. Canale, *Numerical Methods for Engineers*, McGraw-Hill, Inc. 1988.
- [8] J.C. Sarker, *M.S. Thesis*, Electrical Engineering Department, University of Idaho, 1990.
- [9] F. Ali and A. Gupta, *HEMTs and HBTs: Devices, Fabrications, and Circuits*, Artech House, Inc. 1991.



# Formal Specification of a High Speed CMOS Correlator

P. J. Windley  
Department of Computer Science  
University of Idaho  
Moscow, ID 83843  
208.885.6501

**Abstract:** The formal specification of a high speed CMOS correlator is presented. The specification gives the high-level behavior of the correlator and provides a clear, unambiguous description of the high-level architecture of the device.

## 1 Introduction.

The use of formal specification in designing VLSI circuits has many benefits. Perhaps the most important result is a clear description of the design's behavior that can be used for communication among design engineers, production engineers, test engineers, technical writers, and, perhaps most importantly, customers. Formal specifications also provide a firm foundation upon which analysis of the circuit design can take place. This analysis has the potential to significantly reduce design errors as well as providing a basis for demonstrating that the design has desired properties.

This paper presents the formal specification of a high-speed CMOS correlator [2]. The correlator, which is designed to be used in a space-born spectrometer, contains 32 channels and is capable of sampling at 25MHz.

## 2 Formal Specification and Verification.

VLSI devices can be specified at many levels of abstraction [8]. Generally, we need at least a behavioral and a structural specification [4]. The behavioral specification is written in logic and unambiguously describes the expected behavior of the device. The behavioral specification is declarative rather than imperative, giving a clear relationship between the inputs, current state, and outputs.

The structural specification describes, again using logic, how the circuit is put together. Ideally, the structural specification can be derived from design information captured by conventional CAD tools or translated from a hardware description language such as VHDL [6].

Verification is nothing more than a mathematical analysis of the behavioral and structural models. Ideally, we would like to show that the intended behavior follows from the structure. This analysis, which is a type of symbolic simulation, can be done by hand or with the aid of mechanical verification tools [5]. These mathematical models can also be used to analytically demonstrate selected behavioral properties for a computer system.

### 3 A Brief Introduction to HOL.

To formally model hardware and to ensure the accuracy of our proofs, we felt that it was necessary to develop the proofs and properties using a mechanical verification system. This prevents proofs from containing logical mistakes, and assures that the foundations on which the work is based are sound. Due to the nature of the proofs, which include quantification over sets of objects, we felt that a system which supports higher-order logic and a typed lambda calculus would facilitate our efforts. The HOL system was selected for this project due to its support for higher-order logic, generic specifications and polymorphic type constructs. Furthermore its availability, ruggedness, local support, and a growing world-wide user base made it a very attractive selection. In this section we will provide a brief description of HOL.

HOL is a general theorem proving system developed at the University of Cambridge [5,1] that is based on Church's theory of simple types, or higher-order logic [3]. Although Church developed higher-order logic as a foundation for mathematics, it can be used for reasoning about computational systems of all kinds. Similar to predicate logic in allowing quantification over variables, higher-order logic also allows quantification over predicates and functions thus permitting more general systems to be described.

HOL is not a fully automated theorem prover but is more than simply a proof checker, falling somewhere between these two extremes. HOL has several features that contribute to its use as a verification environment:

1. Several built-in theories, including booleans, individuals, numbers, products, sums, lists, and trees. These theories build on the five axioms that form the basis of higher-order logic to derive a large number of theorems that follow from them.
2. Rules of inference for higher-order logic. These rules contain not only the eight basic rules of inference from higher-order logic, but also a large body of *derived* inference rules that allow proofs to proceed using larger steps. The HOL system has rules that implement the standard introduction and elimination rules for Predicate Calculus as well as specialized rules for rewriting terms.
3. A large collection of tactics to support goal directed proof. Examples of tactics include `REWRITE_TAC` which rewrites a goal according to some previously proven theorem or definition, `GEN_TAC` which removes unnecessary universally quantified variables from the front of a goal, and `EQ_TAC` which says that to show two things are equivalent, we should show that they imply each other.
4. A proof management system that keeps track of the state of an interactive proof session.
5. A metalanguage, ML, for programming and extending the theorem prover. Using the metalanguage, tactics can be put together to form more powerful tactics, new tactics can be written, and theorems can be aggregated to form new theories for later use. The metalanguage makes the verification system extremely flexible.



Operator	Application	Meaning
=	$t1 = t2$	$t1$ equals $t2$
,	$t1, t2$	the pair $t1$ and $t2$
$\wedge$	$t1 \wedge t2$	$t1$ and $t2$
$\vee$	$t1 \vee t2$	$t1$ or $t2$
$\Rightarrow$	$t1 \Rightarrow t2$	$t1$ implies $t2$

Table 1: HOL Infix Operators

Binder	Application	Meaning
$\forall$	$\forall x. t$	for all $x, t$
$\exists$	$\exists x. t$	there exists an $x$ such that $t$
$\varepsilon$	$\varepsilon x. t$	choose an $x$ such that $t$ is true

Table 2: HOL Binders

In the HOL system there are several predefined constants which can belong to two special syntactic classes. Constants of arity 2 can be declared to be infix. Infix operators are written " $rand1$  op  $rand2$ " instead of in the usual prefix form: " $op$   $rand1$   $rand2$ ". Table 1 shows several of HOL's built-in infix operators.

Constants can also belong another special class called binders. A familiar example of a binder is  $\forall$ . If  $c$  is a binder, then the term " $c$   $x. t$ " (where  $x$  is a variable) is written as shorthand for the term " $c(\lambda x. t)$ ". Table 2 shows several of HOL's built-in binders.

In addition to the infix constants and binders, HOL has a conditional statement that is written  $a \rightarrow b \mid c$ , meaning "if  $a$ , then  $b$ , else  $c$ ."

## 4 The Correlator Design.

The correlator is designed for a space borne spectrometer. The design accepts two 2-bit data streams clocked at a maximum of 25MHz. Delayed versions of one stream are multiplied (using a biased multiplication) with the undelayed signal on the other stream. The products are accumulated. The process continues for the duration of the integration cycle which is defined by the `int` control line. When the end of an integration period is signaled, the results are latched into a register, the accumulators are cleared, and the `datardy` line goes high to signal that data is ready to be read from the chip. A new integration cycle can begin immediately. Concurrent with the new integration period, the data from the previous integration period can be read on the output lines. Data is read in either a word serial or byte serial mode depending on the value of a control line.

Readers interested in additional detail are referred to [2].

## 5 The Correlator Specification.

This section presents the behavioral specification of the correlator.

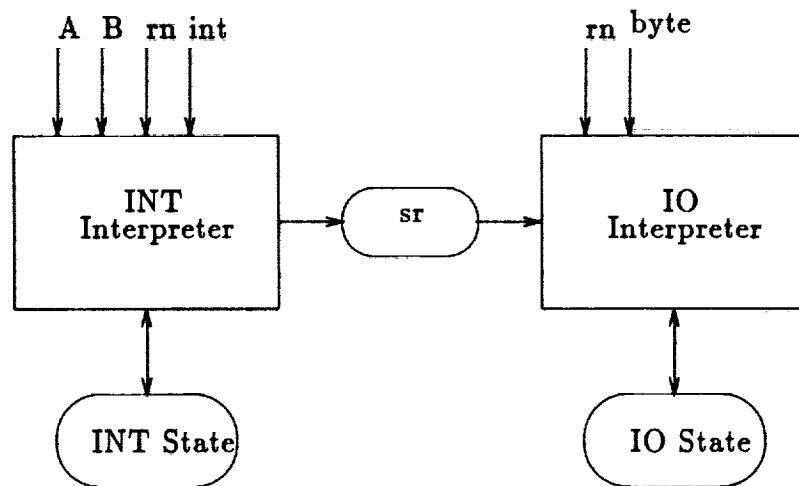


Figure 1: Architecture of the correlator shows the producer—consumer relationship between the INT interpreter and the IO interpreter.

The overall architecture of the behavioral description is shown in Figure 1. The architecture is based on two separate state machines which, along with the datapath, function as single instruction interpreters [7]. The interpreters are arranged in a producer—consumer architecture with a register serving as the shared link between the two interpreters.

The producer portion of the design is the INT interpreter. INT performs the integration of the incoming signals in 32 channels. The interpreter controls the following state variables:

- **acc**—A bank of 32, 4-bit accumulators.
- **delay**—A bank of 32, 2-bit delay elements.
- **sr**—A bank of 32, 24-bit shift registers.
- **count**—A bank of 32, 24-bit counters.

Each of these state variables is parameterized for time and channel number and has type  $\text{time} \rightarrow \text{num} \rightarrow w$ , where  $w$  varies with register width.

The specification for INT relates the state variables at time  $t + 1$  to their value at time  $t$  and the value of the inputs at time  $t$ .

```

 $\vdash_{def}$  integrate_int (acc, delay, sr, count, datardy)
    (a, b, int, rn) =
 $\forall t.$ 
    let nextstate = ((int t)  $\rightarrow$  integrate | dump) = (
    (acc (t+1), delay (t+1), sr (t+1), count (t+1), datardy (t+1)) =
    nextstate (acc t, delay t, sr t, count t, datardy t)
    (a t, b t, int t, rn t))
  
```

The function `nextstate` evaluates to either `integrate` or `dump` depending on the value of the `int` line.

The individual instructions produce new values for the state variables. In the case of the `integrate` instruction new values are calculated for the `acc`, `delay`, and `count` variables. The shift register (`sr`) is unchanged.

```

 $\vdash_{def}$  integrate (acc, delay, sr, count, datardy)
      (a, b, int, rn) =
      let signal_product n = mapper (delay n) b in (
      let new_acc n =
        rn  $\rightarrow$  (bt4_ival 0) |
        (add4 (signal_product n, acc n)) in
      let new_delay n = (n=0)  $\rightarrow$  a | (delay (n-1)) in
      let new_count n =
        rn  $\rightarrow$  (wordn 0) |
        (carry4 (signal_product n, acc n))  $\rightarrow$  inc (count n) |
        (count n) in
      (new_acc, new_delay, sr, new_count, datardy)

```

The new values are precisely described. For example, the new value of the  $n^{th}$  accumulator is calculated by adding a biased multiplication of the  $n$ -delayed signal and the undelayed signal to the current value in the same accumulator.

The consumer portion of the circuit is the I/O interpreter. The interpreter controls the following state variables:

- **sr**—A bank of 32, 24-bit shift registers. This is the same register as the `sr` register in the INT interpreter.
- **counter**—A 7-bit counter for counting the output.
- **out**—A 16-bit register that latches the values on the output lines.
- **borw**—A state variable that indicates whether output is byte or word serial.

The specification for the I/O interpreter is similar to the specification of the INT interpreter. The I/O interpreter has six instructions. The interpreter can be reset, it can start the read cycle, it can end the read cycle, it can dump data from the output registers a byte at a time, it can dump data a word at a time, or it can do nothing.

```

 $\vdash_{def}$  io_int (sr, counter, out, borw, datardy, begin)
  (byte, rn, outck) =
     $\forall t$  .
      let nextstate =
        ((rn t)  $\rightarrow$  reset |
         ((datardy t)  $\wedge$  begin  $\rightarrow$  start_read |
         ((datardy t)  $\wedge$  ((val (counter t)) = 0))  $\rightarrow$  end_read |
         ((datardy t)  $\wedge$  (borw t)  $\wedge$  (outck t)  $\rightarrow$  dump_byte |
         ((datardy t)  $\wedge$   $\neg$ (borw t)  $\wedge$  (outck t)  $\rightarrow$  dump_word |
         noop) in (
        (sr (t+1), counter (t+1), out (t+1), borw (t+1), datardy (t+1)) =
        nextstate (sr t, counter t, out t, borw t, datardy t, begin t)
        (byte t, rn t, outck t))

```

The operation of I0 is more complicated than the operation of INT. Whenever the reset line is raised, the state is reset as described in the specification of the reset operation. When the datardy line goes high, the interpreter begins a read cycle. When the outck line is raised and the datardy line is high, we dump either bytes or words depending on the value of the borw line. There is a counter so that the correct number of bytes and words are dumped. When the counter reaches 0 we end the read cycle (by pulling the datardy line low). Otherwise, we do nothing.<sup>1</sup>

As an example of the instructions in I0, consider the dump\_word instruction.

```

 $\vdash_{def}$  dump_word (sr, counter, out, borw, datardy, begin)
  (byte, rn, outck) =
    let new_counter = (dec counter) in
    let i = (val counter) in
    let new_out = short (sr i) in
    (sr, new_counter, new_out, borw, datardy, begin)

```

The instruction updates the counter by decrementing the old value. The value on the output is determined by 16 most significant bits from the  $i^{th}$  shift register, where  $i$  is the value of the counter.

The most interesting feature of the specification of INT and I0 is that they share state. For example, both specify changes to sr, the variable representing the shift register. INT produces a value that is placed in sr by its dump instruction. I0 uses that value when asked to present the results of the integration on the output lines.

Both interpreters also specify changes to datardy, the variable representing whether or not data is ready to be output. INT sets datardy when it has dumped the contents of the counter into the shift register. I0 resets datardy when it is done outputting the data.

Readers of this specification who are familiar with the design may be surprised to find that some details in the circuit are not found in the specification. For instance, after the end of the integration period ends, there is an 8 cycle delay before data can be read from

<sup>1</sup>Note that count in INT and counter in I0 are two different state variables.

the chip (i.e. `datardy` goes high). In the specification shown above `datardy` goes high the time period after the `int` line is pulled high. This is an example of the temporal abstraction going on between the circuit levels of the specification and the behavioral specifications given here.

## 6 The Top-Level Specification

The final specification combines the specifications of the two interpreters and operates them in parallel.

```

 $\vdash_{def}$  corr_top rep (acc, delay, sr, count, datardy,
                    begin, counter, out, borw)
                    (byte_e, rn, outck, a, b, int) =
                    ((integrate_int rep (acc, delay, sr, count, datardy)
                      (a, b, int, rn))  $\wedge$ 
                     (io_int rep (sr, counter, out, borw, datardy, begin)
                      (byte_e, rn, outck)))

```

The specification does not explicitly answer questions regarding the shared use of the `sr` and `datardy` lines. For example, do INT and IO correctly coordinate the writing and reading of `sr` correctly? This and other important questions regarding the operation of the correlator can be answered by analysis of the specification.

## 7 Conclusion.

This paper has presented the behavioral specification for a VLSI correlator design. Previous to this specification being written, the design was described in design documents and papers such as [2]. These descriptions were necessarily ambiguous since they were written in English. Deriving the specification by reading the design documents and talking to the design engineer provides an interesting perspective on the design process. The behavioral specification of the correlator documents the design and is useful for enhancing communication between designers, customers, and users by unambiguously describing the function of the device.

The specification presented in this paper is a snapshot of the design. A specification is constantly subject to revision to bring it up to date with current expectation and to correct errors that are part of any written description. Future work will extend the specification in two ways:

- We intend to show that the specification meets certain requirements for correct operation. For example, the analysis will make explicit the synchronization conditions that must exist between the two interpreters for the chip to function correctly and show that they are met.

- We will specify the structural level by deriving it from the design information captured in the HDL description of the circuit. We intend to show that this structural specification implies the architecture we have described above.

## Acknowledgments

This work was sponsored by NASA under Space Engineering Research Center grant NAGW-1406.

## References

- [1] Albert Camilleri, Mike Gordon, and Tom Melham. Hardware verification using higher order logic. In D. Borriane, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*. Elsevier Scientific Publishers, 1987.
- [2] J. Canaris and S. Whitaker. A high speed CMOS correlator. In *NASA Space Engineering Research Center Symposium on VLSI Design*, pages 3.3.1–3.3.11, November 1990.
- [3] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5, 1940.
- [4] Michael J.C. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. Elsevier Scientific Publishers, 1986.
- [5] Michael J.C. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwhistle and P.A Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*. Kluwer Academic Press, 1988.
- [6] IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual*, 1987.
- [7] Phillip J. Windley. *The Formal Verification of Generic Interpreters*. PhD thesis, University of California, Davis, Division of Computer Science, June 1990.
- [8] Phillip J. Windley. A hierarchical methodology for the verification of microprogrammed microprocessors. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1990.

## **A Verification Logic Representation of Indeterministic Signal States**

J. W. Gambles and P. J. Windley

NASA Space Engineering Research Center for VLSI Systems Design

University of Idaho

Moscow, Idaho 83843

***Abstract*** - The integration of modern CAD tools with formal verification environments require translation from hardware description language to verification logic. A signal representation including both unknown state and a degree of strength indeterminacy is essential for the correct modeling of many VLSI circuit designs. A higher-order logic theory of indeterministic logic signals is presented.

### **1 Introduction**

As higher transistor counts increase the complexity of VLSI circuits and the number of potential test cases explode, formal verification methods promise value in design fault exclusion. Before verification is accepted by design engineers, stand alone verification tools that are used in the academic research arena must be integrated with the CAD tools being used by VLSI designers. One major benefit of this integration is that VLSI designers will enjoy increased confidence that abstract behavioral models are correct. There are several reasons a VLSI designer may choose to use abstract behavioral models. In a top-down design, a behavioral description may be used to simplify circuit understanding before the implementation is designed. A behavioral model can be utilized as part of a simulation of the entire system at an early date. After the circuit structure is designed and modeled, the logic simulation of complex systems can become very slow. The simulation can be made faster by replacing circuit blocks with the corresponding behavioral model. The problem with these design approaches is that there is currently no way to relate the circuit structural model to the abstract behavioral model. Having a verification tool available in the VLSI CAD tool suite would allow these models to be related through mathematical analysis.

The hardware description languages (HDL) used by VLSI CAD tools can provide the link between these tools and the verification environment. Engineers can design using the CAD tool HDL and this description can be automatically translated for use in the verification tool. This paper examines the translation of logic signal representations from the BOLT (Block Oriented Logic Translator) HDL, used in the NOVA simulation engine, to the HOL theorem proving system.

## 2 HOL

HOL is a general theorem proving system developed at the University of Cambridge [4,6] based on Church's theory of simple types, or higher-order logic. Higher-order logic is suitable for specifying all aspects of hardware, including both structure and behavior [6,8]. In using higher-order logic, predicates are defined to represent both circuit primitives and behavioral definitions [4]. First-order logic is well suited to represent simple combinational circuits, but not sequential circuits. In higher-order logic, variables are allowed to range over functions and predicates which make it suitable for representing sequential circuit behavior [8]. HOL is not an automated theorem prover but is more than simply a proof checker, falling somewhere between these two extremes. Translation from BOLT descriptions to HOL predicates requires that HOL primitives be defined to correspond to the BOLT circuit representations.

Symbols in HOL are represented by strings of ASCII characters. Conjunction, disjunction, negation, implication, and equality are represented by  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\Rightarrow$ , and  $=$  respectively. Universal quantification (for all) is symbolized  $!$  and existential quantification (there exists) is  $?$ . The function composition operator is  $\circ$  and the conditional expression "if  $a$  then  $b$  else  $c$ " is symbolized  $a \Rightarrow b \mid c$ .

## 3 Logic States and Strengths

Few modern VLSI circuits are designed using only classical logic gates [3,10]. In designs using pass-transistor, tri-state, and pre-charge logic, it is common for circuit nodes to be driven from multiple circuit elements. These multiple drivers are designed to have differing drive strengths in order for one to dominate over another in cases of contention. The drive strength can be considered to be closely related to current drive (charge sourcing) capability [7,2]. The signal values represented in the NOVA simulation engine are an extension of Bryant's lattice theoretic approach [7,11]. In the lattice theoretic approach the elements in the domain of signal values represent the combination of logic state, from the set **True**, **False**, and **Unknown**; and a signal strength. These signal values form a partially ordered set with their order based on strength dominance when circuit output values are combined.

While Bryant later abandoned the lattice theoretic approach [2] stating "while this approach at first seems very elegant, it cannot adequately describe the effects of transistors in the X (Unknown) state," Cameron and Shovic have shown that the problem with the **Unknown** state can be corrected by extending the domain of signal values to include some degree of strength indeterminacy [3]. Thus, the signal values are extended to represent both logic states and a range of signal strength.

The **Unknown** state can be the result of a node connected to two drivers, one driving to a **True** and the other driving to a **False**, neither driver having sufficient strength to dominate the other; or simply a node whose voltage is not yet known. Combining the cases of "invalid" logic level and "valid but not known" into a single **Unknown** state simplifies the simulation algorithm but may make the simulator pessimistic since it will propagate



the Unknown state when resolving some circuit nodes[2].

We refer to the combination of state and strength information as STATES. The STATES representation presented here is consistent with that presented in [3,10] except the total number of strengths  $N$ , is extended to include a weakest strength, Nil, which represents a node that is disconnected from all charge sources. By definition, a signal being driven by the Nil strength must be at the Unknown state.

### 3.1 Representation of STATES

Given the set of states True, False, and Unknown and a fully ordered set of strengths  $\sigma_1, \sigma_2, \dots$ , and  $\sigma_N$  we can define STATES. The STATES corresponding to the states True and False are represented as a triple **Kbd** where:

**K** is 1 or 0 representing the logic state True or False;

**bd** represents a indeterminate *range* of strengths where:

**b** is the strongest possible strength ( $\sigma_1 \leq \mathbf{b} \leq \sigma_{N-1}$ ) which sets a lower bound on the strength of a signal that can overdrive this state;

**d** is the weakest possible strength ( $\mathbf{b} \leq \mathbf{d} \leq \sigma_{N-1}$ ) which sets a upper bound on the strength of a signal that this state can overdrive.

The STATES corresponding to the Unknown state are represented as a triple **Xpq** where:

**X** represents the Unknown state;

**p** is the strongest possible strength driving toward 0 ( $\sigma_1 \leq \mathbf{p} \leq \sigma_{N-1}$ ) which sets a lower bound on the strength of a signal that can overdrive this state to a 1;

**q** is the strongest possible strength driving toward 1 ( $\sigma_1 \leq \mathbf{q} \leq \sigma_{N-1}$ ) which sets a lower bound on the strength of a signal that can overdrive this state to a 0.

### 3.2 The Number of STATES

For  $N$  strengths the number of True and False STATES is:

$$TF\_STATES(N) = 2((N-1) + (N-2) + \dots + 1) = (N-1)(N) \quad (1)$$

For the Unknown state the number of STATES is:

$$X\_STATES(N) = (N-1)^2 + 1 \quad (2)$$

The plus one term in equation (2) represents the combination of Unknown state and weakest strength,  $\sigma_N = Nil$ . This STATE is referred to as Nil. Thus, the total number of STATES for  $N$  strengths is equal to:

$$TOTAL\_STATES(N) = 2N^2 - 3N + 2 \quad (3)$$

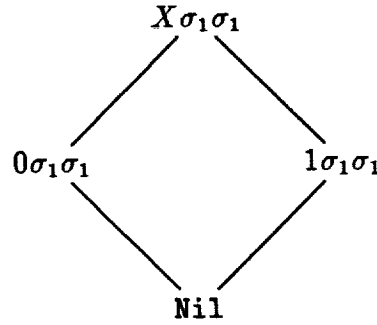


Figure 1: Base Case Signal Lattice (N=2)

## 4 STATES Theory

A complicated algorithm for determining the result of combining STATES is presented in [3]. This algorithm is not satisfactory for use in HOL. We have developed a lattice that describes the result of joining two signals. In this lattice theoretic approach to signal strengths, the *join* (least upper bound) operation represents the resolution of contending circuit elements [11].

The lattice structure is described through the notion of immediate superiors or *covers*. For two elements,  $a$  and  $b$  of a partially ordered set,  $a$  *covers*  $b$  if and only if  $a > b$  and there exists no element  $x$  of the partially ordered set such that  $a > x > b$  [1]. A list of all of the elements and covers completely describe a lattice. The covers can also be used to define a graph of the lattice. The vertices of the graph are the elements and the segments of the graph represent the covers. If the graph is drawn such that whenever  $x$  covers  $y$ , the vertex  $x$  is *higher* than the vertex  $y$ , it is called a "Hasse diagram" of the lattice [1].

### 4.1 Defining STATES Lattice Structure

Given the base case  $N = 2$  ( $N = 1$  is a trivial case of one single STATE, Nil) there are four STATES and no strength indeterminacy, meaning there is only a single value ( $\sigma_1$ ) within the range of possible strengths. There are four covers and the lattice Hasse diagram is as presented in [7,11], a simple diamond (Figure 1).

To extend a  $N$  strength Hasse diagram (lattice) to  $N + 1$  strengths:

1. Add three STATES and four covers to form a new diamond at the bottom of the  $N$  strength diagram by replacing Nil with  $X\sigma_N\sigma_N$ , adding  $0\sigma_N\sigma_N$  and  $1\sigma_N\sigma_N$  each covered by  $X\sigma_N\sigma_N$  and placing Nil at the bottom of the diagram covered by both  $0\sigma_N\sigma_N$  and  $1\sigma_N\sigma_N$ .
2. For each  $M = N$  to 2, by -1, add the following STATES and covers:

- (a)  $X\sigma_{M-1}\sigma_N$  covered by  $0\sigma_{M-1}\sigma_{N-1}$  and covering  $X\sigma_M\sigma_N$
- (b)  $X\sigma_N\sigma_{M-1}$  covered by  $1\sigma_{M-1}\sigma_{N-1}$  and covering  $X\sigma_M\sigma_N$
- (c)  $0\sigma_{M-1}\sigma_N$  covered by  $X\sigma_{M-1}\sigma_N$  and covering  $0\sigma_M\sigma_N$
- (d)  $1\sigma_{M-1}\sigma_N$  covered by  $X\sigma_N\sigma_{M-1}$  and covering  $1\sigma_M\sigma_N$

## 4.2 The Number of Covers

The total number of covers for  $N$  strengths is equal to:

$$COVERS(N) = 4N^2 - 10N + 8 \quad (4)$$

## 4.3 The Lattice Structure for NOVA

The NOVA simulation engine and BOLT HDL have been selected for this research so that we may have access to commercial-scale designs written by nonacademic VLSI designers while a translation tool to HOL is developed. In NOVA,  $N = 4$  and  $\sigma_1 = a$  (active),  $\sigma_2 = r$  (resistive),  $\sigma_3 = f$  (float) and  $\sigma_4 = \text{Nil}$ . Note that float  $>$  Nil and can be used to represent signal levels at charged capacitive nodes. For  $N = 4$ , equation (3) yields 22 STATES and equation (4) yields 32 covers. The Hasse diagram for the STATES and covers for NOVA is shown in figure 2. In addition to identifying the list of covers required to define the lattice structure in the verification logic, the Hasse diagram also provides a quick, visual understanding of the resolution of joined STATES.

## 5 Implementing STATES in HOL

The HOL system includes a type definition package that allows the user to define new types and prove theorems about essential properties of the new type. The type package automatically carries out much of the necessary formal proof required for a new type definition. Theorems about the new type are proven, rather than simply postulating axioms for the new type, in order to avoid the introduction of inconsistency into the logic [9]. A new type for signal values, called **strength**, is defined in HOL by enumeration of all of the STATES. Properties proven about the new type include each value being distinct, an induction theorem, and a case analysis (perfect induction) theorem. The STATES lattice is defined by enumeration of the covers and the function **join** is defined to be the least upper bound. Once the join function definition is complete, consistency of proofs that utilize **join** are insured by formal proof of the lattice theoretic obligations [11] for the join operation. These obligations are:

1. Idempotence. For all  $a$  STATES,  $\text{join } a \ a = a$ .
2. Commutativity. For all  $a$  and  $b$  STATES,  $\text{join } a \ b = \text{join } b \ a$ .
3. Associativity. For all  $a$ ,  $b$  and  $c$  STATES,  $\text{join } a \ (\text{join } b \ c) = \text{join } (\text{join } a \ b) \ c$ .

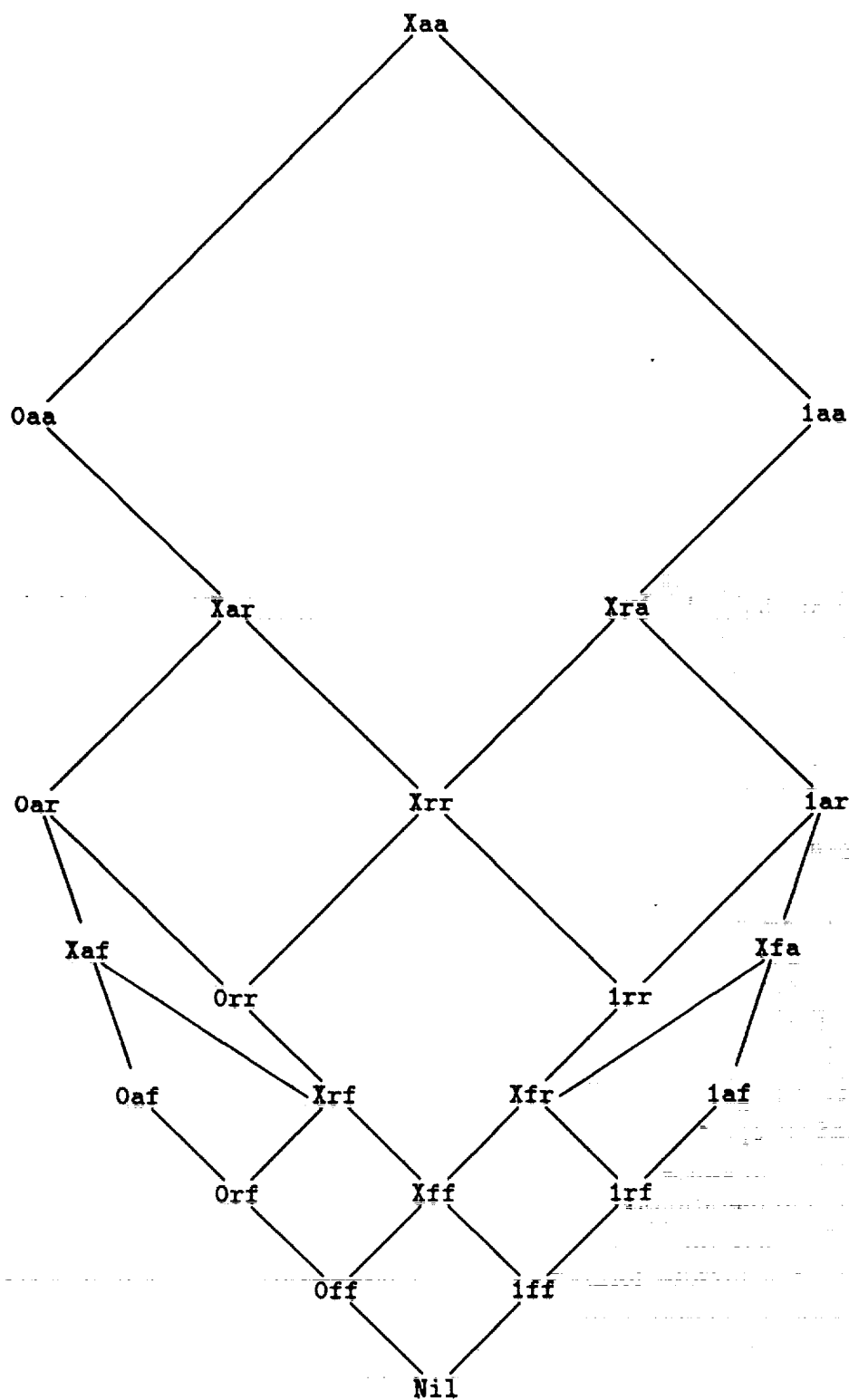


Figure 2: Signal Lattice for N=4 (NOVA)

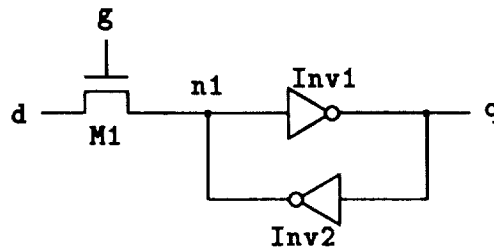


Figure 3: Memory Cell Schematic Diagram

4. Existence of bottom. For all  $a \text{ STATES}$ ,  $\text{join } a \text{ Nil} = a$ .

### 5.1 STATES Abstraction Function

Typically a behavioral specification is defined in terms of boolean values. An abstraction function is required to relate **STATES**, used in structural specifications, to boolean values.

```
STATES_ABS sig = ((sig=1aa)\/(sig=lar)\/(sig=lrr)\/
                  (sig=1af)\/(sig=lrf)\/(sig=lff)) => T |
                  ((sig=0aa)\/(sig=0ar)\/(sig=0rr)\/
                  (sig=0af)\/(sig=0rf)\/(sig=0ff)) => F |
                  ARB
```

The Unknown **STATES** are assigned a value **ARB**, defined to be an arbitrarily chosen boolean value.

## 6 Theory Demonstration

A static memory circuit cell, implemented with gate level and pass transistor primitives, is used to demonstrate the **STATES** theory (Figure 3). Without a signal value representation that realizes output dominance this circuit cannot be correctly modeled. Fundamental to the operation of this circuit is that the output strength of pass-transistor **M1** dominates the output of inverter **Inv2** to force node **n1** to the state of the input **d** while the gate **g** is **True** (high voltage). The feedback inverter **Inv2** acts to store the state, by dominating the pass-transistor after the gate goes **False**, turning the transistor off.

### 6.1 The Circuit Primitives

The memory cell structure includes three predicate definitions; a pass-transistor element, inverter elements, and the **JOIN** operation. Time is represented as a number (**num**) stream and circuit signals are defined to be functions of type **num** to type **strength**.

The behavioral model of the cell is not defined for the gate input being at an unknown state. A simplified pass-transistor model is used that defines that the signal at the drain is equal to the signal at the source if the gate is `True`, else it is `Nil`.

```
NTRAN (g,s,d) =
! t.
  d t = ((g t =laa)\/(g t =lar)\/
        (g t =lrr)\/(g t =laf)\/
        (g t =lrf)\/(g t =lff)) => s t |
        Nil
```

The inverter predicate has five arguments. The first three arguments are of type `strength` and define the possible inverter output STATES. The first is the output STATES for a `True` output, the second for a `False` output, and the third the `Unknown` state output. The `Unknown` output value is derived from the strongest `True` and `False` strengths. The fourth and fifth arguments are signal functions of type `num` to type `strength`. The fourth is the inverter input and the fifth is the output.

```
INV ls Os Xs (in,out) =
! t.
  out t =(((in t =laa)\/(in t =lar)\/
           (in t =lrr)\/(in t =laf)\/
           (in t =lrf)\/(in t =lff)) => Os |
           ((in t =0aa)\/(in t =0ar)\/
           (in t =0rr)\/(in t =0af)\/
           (in t =0rf)\/(in t =0ff)) => ls |
           Xs )
```

## 6.2 JOIN

The JOIN predicate performs two operations. It determines the resulting signal value of combining circuit outputs by applying the join function. The second operation is related to the sequential behavior of a charge storage node. The capacitance of a node may result in a time delay when the node is driven to a new signal level. The delay time increases as the strength of the driving signal decreases. This sequential behavior is modeled as having a variable delay, whose length is based on the strength of the join function result. [5,7]. The Hasse diagram shows the relative strength of STATES and can be used to abstract the delay values for individual STATES by segregating them into horizontal bands on the diagram. All STATES within a common band have the same delay and the delay is longer for lower bands. For cases where it is desired to model different delays for rise and fall times the diagram can be segregated right from left also.

The demonstration cell is modeled as having two possible delays. When the pass-transistor is turned on, the storage node at the join is driven by an active strength and the delay is defined to be zero. When the pass-transistor is turned off, the storage node

is driven by the resistive strength of the feed-back inverter and the delay is defined to be one.

```
JOIN (s',s'',s:num->strength) =
  ! t. let sig = join (s' t) (s'' t) in
    ((sig = 0aa) /\
     (sig = 1aa) /\
     (sig = Xaa) /\
     (sig = Xar) /\
     (sig = Xra)) => (s t = sig)      |
                      (s (t+1) = sig)
```

### 6.3 The Structural Description

A BOLT description of the cell is:

```
MODULE Q .CELL G D;

BEGIN
  N1 .NTRAN G D;
  Q .INVR N1;
  N1 .INVR Q (STR='RR');
END;
```

The STR='RR' parameter in the second INVR invocation defines the output strength of that inverter as resistive. The default value used for the first invocation is active. The HOL structural specification of the cell is:

```
cell_IMP (d,g,q) =
  ? n1 n1' n1'':num->strength .
    NTRAN (g,d,n1')          /\
    INV 1aa 0aa Xaa (n1,q)    /\
    INV 1rr 0rr Xrr (q,n1'') /\
    JOIN (n1',n1'',n1)
```

### 6.4 The Behavioral Description

When the gate of the pass-transistor is *True* the cell is *writing* the input and the output, q, follows as the inverse of d. When the gate is *False* the cell is *storing* the previous data. The HOL behavioral description is:

```

cell_SPEC (d,g,q) =
  ! t.
    (g t) => (q t = ~d t)    |
              (q (t+1) = q t)

```

## 6.5 The Cell Verification

Because the operation of the cell requires that the output of the pass-transistor dominate the resistive strength output of INV2 and the pass-transistor is not an amplifier, there is a validity condition that the signal applied to input d must be stronger than resistive. This condition is required for proper circuit operation and is not simply a verification artifact.

```

Valid1 (d) =
  ! t.
    (d t = laa) \/\ (d t = 0aa)

```

Because the behavior of the cell is defined only for boolean value signals at the gate, there is a validity condition for the gate that it be either a True or False state. This condition yields a 12 way case analysis in the proof, but is easily reduced to needing to consider only the two cases of writing and storing.

```

Valid2 (g) =
  ! t.
    (g t = laa) \/\ (g t = lar) \/\ (g t = lrr) \/\
    (g t = laf) \/\ (g t = lrf) \/\ (g t = lff) \/\
    (g t = 0aa) \/\ (g t = 0ar) \/\ (g t = 0rr) \/\
    (g t = 0af) \/\ (g t = 0rf) \/\ (g t = 0ff)

```

The verification of the cell entails proving that the cell structural description and validity conditions logically imply the behavioral specification. The theorem proven is:

```

|- (Valid1 (d) /\ Valid2 (g) /\ cell_IMP(d,g,q)) ==>
    cell_SPEC(STATES_ABS o d, STATES_ABS o g, STATES_ABS o q)

```

## 7 Future Work

The theory of signal lattices presented in this paper is an important first step in linking BOLT and HOL. Future steps include:

1. Developing and validating a set of HOL theories corresponding to the primitive components in the NOVA library.
2. Writing a formal semantics for BOLT.
3. Embedding BOLT's formal semantics in HOL.

These steps do not include work on translating NOVA behavioral models to HOL, a difficult, but necessary task.



## 8 Conclusion

The first step in the integration of CAD VLSI design tools with a verification tool is the translation of the HDL representations into the verification logic. A verification logic theory has been presented for reasoning about an indeterministic signal value representation based on a lattice approach. This work is necessary because the previous algorithm for joining indeterministic signal values is not suitable for a verification logic environment. The suitability of the lattice approach is demonstrated through the verification of a static memory cell. The lattice diagram presented also quickly provides to users the result of combining different valued indeterminate signals.

## 9 Acknowledgements

This research was supported by NASA under Space Engineering Research grant NAGW-1406.

## References

- [1] Birkhoff, G., *Lattice Theory*, American Mathematical Society, 1948.
- [2] Bryant, R. E., "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, Vol. C-33, pp.160-177, February 1984.
- [3] Cameron, K. B. and Shovic, J. C., "Calculating Minimum Logic State Requirements for Multi-Strength Multi-Value MOS Logic Simulators, " 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors, IEEE Computer Society Press, pp. 672-675, 1987.
- [4] Camilleri, A., Gordon, M. and Melham, T., "Hardware Verification Using Higher Order Logic," in D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, Elsevier Scientific Publishers, pp. 43-67, 1987. Also Technical Report No. 91, University of Cambridge Computer Laboratory, September 1986.
- [5] Dhingra, I. S., "Formal Validation of An Integrated Circuit Design Style," in G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, Kluwer Academic Publishers, pp. 293-321, 1988. Also Technical Report No. 115, University of Cambridge Computer Laboratory, August, 1987.
- [6] Gordon, M. J. C., "HOL: A Proof Generating System for Higher-Order Logic," in G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, Kluwer Academic Publishers, pp. 73-128, 1988. Also Technical Report No. 103, University of Cambridge Computer Laboratory, August, 1987.
- [7] Hayes, J. P., "A Unified Switching Theory with Applications to VLSI Design," *Proceedings of the IEEE*, Vol. 70, No. 10, pp.1140-1151, October 1982.

- [8] Melham, T. F., "Abstraction Mechanisms for Hardware Verification," in G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, Kluwer Academic Publishers, pp. 267-291, 1988. Also Technical Report No. 106, University of Cambridge Computer Laboratory, May, 1987.
- [9] Melham, T. F., "Using Recursive Types to Reason About Hardware Verification," Technical Report No. 135, University of Cambridge Computer Laboratory, May, 1988.
- [10] Miles, L., Prins, P., Cameron, K., and Shovic, J., "NOVA: A New Multi-Level Logic Simulator," 2nd NASA SERC Symposium on VLSI Design, pp. 4.1.1-4.1.13, 1990.
- [11] Ullman, J. D., *Computational Aspects of VLSI*, Computer Science Press, 1984.

## Formal Verification of State Machines

M. Alahmad and P. Windley  
NASA Space Engineering Research Center  
for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

**Abstract** – A formal specification of VLSI state machines based on a sequence invariant architecture is presented. The behavioral description represents a logical description of any synchronous state machine. The structural specification represents an adoptive architecture developed using VLSI technology to implement the state machine. This specification becomes a tool for future verification and specification of state machines using dedicated machines and/or alternative technologies. The verification of the state machine is done in HOL, a theorem proving system. Using HOL, the verification shows analytically that the circuit structure has the desired behavior.

### 1 Introduction

With the advancement of integrated circuit technology, the need for new methods of ensuring design correctness is becoming more prominent. Simulation remains the dominant method in use, but, recently, interest has grown in using formal logical analysis to show the correctness of digital systems.

Formal verification of hardware involves using theorem-proving techniques to verify that a stated behavioral definition of a circuit is a logical consequence of the structural description of the circuit, i.e., proving that the structure of the circuit forces it to behave as stated. This paper presents a formal specification and verification of a general state machine. The specification describes the behavior and structure of the state machine. The behavioral specification is a logical representation of a state machine. Using a particular design in VLSI technology, a structural description based on the Sequence Invariant Architecture is described. The structure clearly specifies how components are connected and built to achieve the operation of the state machine. The verification shows, by analysis, that the structural specification implies the behavioral specification using a theorem proving system known as HOL [1]. Hence, the VLSI architecture is capable of implementing any state machine.

### 2 The HOL System

As described by Birtwistle and Subrahmanyam [3], the HOL system ('HOL' standing for 'higher order logic') is designed to facilitate the interactive generation of formal proofs. A logic in which problems can be expressed is interfaced to a *programming language* in which

proof procedures and strategies can be encoded. The combination enables deduction in logic (in the sense of chains of primitive inference steps) to be produced by invocation of programming constructs at a higher level of abstractness.

The logic part of HOL is conventional higher-order logic. New types, constants and axioms can be introduced by the user, and organized in logic *theories*. The programming language of HOL is ML (for 'meta-language'). The *type discipline* of ML ensures that the only way to create *theorems* in the object logic is by performing proofs; theorems have the ML type *thm*, objects of which can only be constructed by the application of interface rules to other theorems or axioms.

### 3 Sequential Circuits Overview

Sequential circuits are categorized as either synchronous or asynchronous, depending upon whether or not the behavior of the circuit is clocked at discrete instants of times. The operation of synchronous sequential circuits (the topic of this paper) is controlled by a synchronizing pulse signal called a clock pulse or simply a clock.

Sequential machines are usually represented by state diagrams or state tables (flow tables). A flow table has a row corresponding to every internal state of the machine and a column corresponding to every possible input. The entry in row  $q_i$  and column  $I_m$  represents the next state produced if  $I_m$  is applied when the machine is in state  $q_i$ . Table 1 shows a flow table for an arbitrary circuit with six-states and three inputs. Once the flow table is constructed for a given circuit, a state assignment is performed. A state assignment is the encoding of the states of the flow table with the internal state variables  $(y_1, y_2, \dots, y_n)$ . Table 2 shows the state assignment and the next state entries assignment for Table 1. Finally, the next state equations are derived from the state assignment using Karnaugh map techniques. We can also derive an equation that describes the output behavior from the flow table.

#### 3.1 SISM Overview

An adaptive hardware architecture has been developed [2], that enables the designer to design any sequential circuit based on the width of the machine  $w$ , and the number of control inputs  $I$ , without a knowledge about the sequence to be incorporated. This adaptive architecture is called a Sequence Invariant State Machine (SISM) design.

With the SISM realization, any flow table can be implemented without a change in the hardware configuration. That is given  $w$ , and  $I$ , a hardware circuit is easily derived, that can implement any state machine that has a maximum of  $I$  control inputs, and  $2^w$  internal states.

#### 3.2 Architecture And Operation

	$I_1$	$I_2$	$I_3$
A	C, 1	B, 1	A, 0
B	D, 0	C, 1	B, 0
C	E, 0	D, 0	C, 0
D	F, 1	E, 1	D, 1
E	A, 0	F, 0	E, 1
F	B, 0	A, 1	F, 1

Table 1: General 6-states, 3-input flow table.

$y_1$	$y_2$	$y_3$		$I_1$	$I_2$	$I_3$
0	0	0	A	0 1 0, 1	0 0 1, 1	0 0 0, 0
0	0	1	B	0 1 1, 0	0 1 0, 1	0 0 1, 0
0	1	0	C	1 0 0, 0	0 1 1, 0	0 1 0, 0
0	1	1	D	1 0 1, 1	1 0 0, 1	0 1 1, 1
1	0	0	E	0 0 0, 0	1 0 1, 0	1 0 0, 1
1	0	1	F	0 0 1, 0	0 0 0, 1	1 0 1, 1
1	1	0	G	0 0 0, 0	0 0 0, 0	0 0 0, 0
1	1	1	H	0 0 0, 0	0 0 0, 0	0 0 0, 0

Table 2: State Assignment for Table 1.

Figure 1 shows a general SISIM architecture, this architecture can be used to implement one of the next state variables in Table 2.

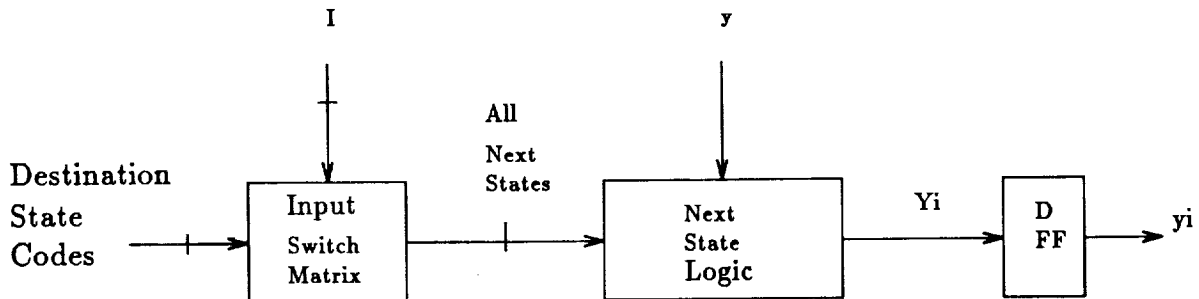


Figure 1: General SISIM Architecture.

The architecture contains the following components:

- The destination state codes are derived from the next state entries in the state assignment table by inspection. For example, the destination state codes for state B and state variable  $y_i$  are the next state bits  $Y_i$  associated with state B. Therefore, the destination state codes for state B are (000, 110, 101) under control inputs ( $I_1; I_2; I_3$ ) and variables ( $y_1; y_2; y_3$ ) respectively. One way to implement those codes is to use constants, that is, presenting ones and zeros at the input of the structure. Also, they could be programmed into the structure using various memory devices [3].

- The input switch matrix is combinational logic that produces all the possible next state entries for each current control input.
- The next state logic consists of an independent path for each of the present states in the state assignment flow table.
- The storage element is a D-FF that preserves the present state.

The operation of the architecture is as follows. The current control input selects the set of potential next states that the circuit can assume (input column in the flow table). The present state variables select the exact next state (row in the flow table) that the circuit will assume at the next clock pulse.

## 4 Formal Specification

The previous section presented a description of the SISM architecture and operation. This section presents the formal specification of the SISM architecture. The behavioral specification is introduced first and then a structural implementation is described.

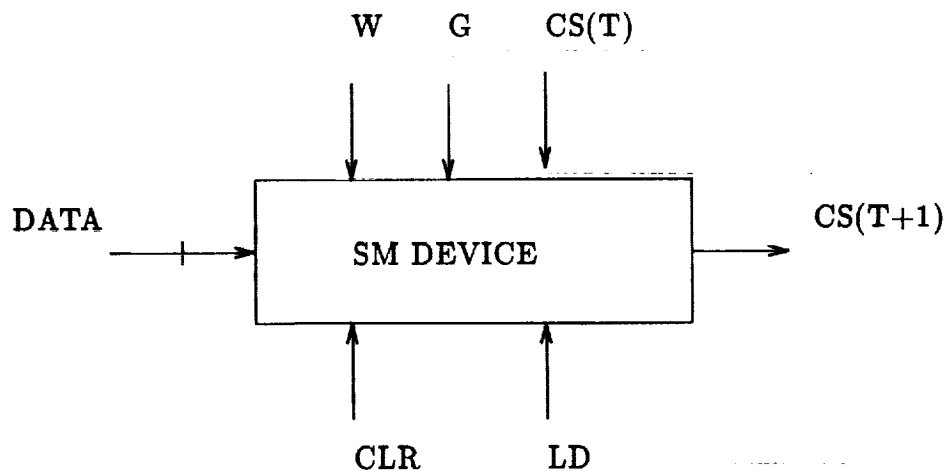


Figure 2: General state machine device

### 4.1 The Behavioral Specification

A general behavioral description of all state machines can be specified by defining a predicate that relates the inputs and outputs and defines the state transition. Figure 2 shows a general state machine device. The behavior of the state machine device can be specified by a predicate `sism-spec`, that is true only when the combination of the values of the variables `w`, `g`, `data`, `clr`, `ld`; and the state variable `cs` is one that could occur on the corresponding input and output signals of the device. The variables are references to actual signals and data as explained below.

- **'w'**, " $(:num)$ ".  
This represents the width of the state machine, i.e., the number of next state variables.
- **'g'**, " $(:time \rightarrow num)$ ".  
This is the control input to the state machine. It is represented as function associated to time. That is at time (t), the input (g) is the control input which is a number from zero to  $I$ . Where  $I$  is the maximum number of control inputs.
- **'data'**, " $(:num \rightarrow num \rightarrow num \rightarrow bool)$ ".  
This is the destination state codes for the entire state machine. It is represented as a function associated with the width of the state machine and the list of data for each of the next state variables.
- **'clr'**, " $(:time \rightarrow bool)$ ".  
This signal when enabled will forces the output values to be cleared to low.
- **'ld'**, " $(:time \rightarrow bool)$ ".  
This signal when enabled will load the input data to the D-ff and present it to the output.
- **'cs'**, " $(:time \rightarrow num \rightarrow bool)$ ".  
This is the current state value. It is represented as function associated to time. That is at time (t) this value will enable one path from the input to the output.

The overall behavior of the state machine is given by the following logic term:

```

sism-spec =
 $\vdash_{def}$  sism_spec w g data clr ld
      (cs:num $\rightarrow$ num $\rightarrow$ bool) =
      ( $\forall$  t:num.   cs (t+1) = (clr t  $\rightarrow$  ZEROS w |
                        ld t  $\rightarrow$  data (g t) (val w (cs t)) |
                        cs t))"
```

The predicates **sism-spec** asserts that the relationship between those values corresponds to the way the state machine works in practice. That is, the next state of the machine at time (t+1) is a function of the value of the data input and the current state at time (t).

## 4.2 The Structural Specification

An implementation of state machines based on the sequence invariant architecture is presented. Using tools available in HOL the structure of the SISM can be described by specifying high level descriptions of the major pieces of the SISM device and combining them so that they correspond to the actual structure. The structure of the SISM can be represented by a predicate **sism-imp** with a definition as follows:

### 10.3.6

```
(sism_imp =
  sism_imp w g data clr ld cs = (sism_imp_rec w w g data clr ld cs)"
```

The predicate `sism-imp-rec` defines the structure of the circuit. The predicate is defined recursively on its width indicating the iterative structure of the circuit. The predicate is defined as follows:

```
(sism_imp_rec =
  "(sism_imp_rec 0 w g data clr ld cs = block 0 w g data
    clr ld cs)
  ^
  (sism_imp_rec (n+1) w g data clr ld cs =
    ((sism_imp_rec n w g data clr ld cs) ^
    (block (n+1) w g data clr ld cs)))"
```

The predicate `block` gives the structure of a single slice of the circuit. `Block` is defined by conjoining the predicates that specify the behaviors of each component with the logical connective ( $\wedge$ ) and using existential quantification ( $\exists$ ) to hide the internal signals. The following logic term describes `block`:

```
block =
  ⊢def block id w g data clr ld cs =
    (∃ out1 out2.
      (sel id w g data out1) ^
      (mux w out1 cs out2) ^
      (d_ff out2 ld clr (cs id)))"
```

In this definition the two internal lines (`out1`; `out2`) are hidden from the external environment using the existential quantifier ( $\exists$ ). The definition of `block` states that the values which can appear on the external inputs and outputs of the SISM device are precisely those which satisfy the constraints imposed by the predicates modeling the three modules from which it is built. The modules that are used to define the predicate `block` are explained next.

**The Selector module** The selector module is defined using predicates as a function. The predicates that defines the behavior specification is a function as shown below,

```
sel=
  ⊢def sel id w g data out =
    ∀ (t:time) line.
      (line < (2 EXP (SUC w))) ⇒
      (out line t) = (data id line (g t))";;
```



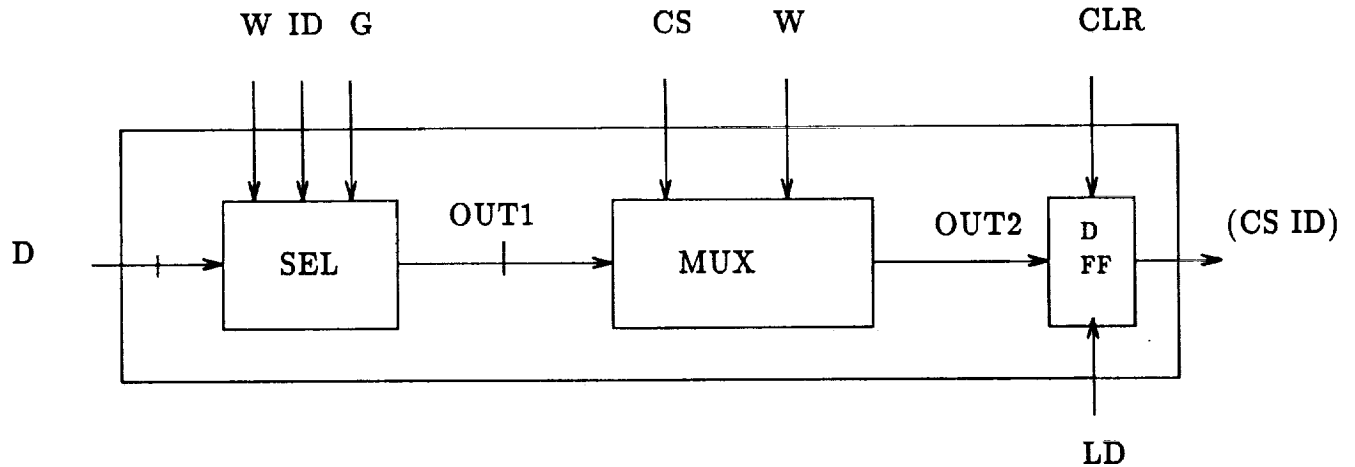


Figure 3: A block representing the SISM device

The selector is a device that is controlled by the control inputs. For each block there are  $2^w$  selectors. Hence,  $2^w$  outputs are presented to the next device. The data input to the selector are the destination state codes. The outputs are all the data selected by the current control input. Referring to the definition and to Figure 3, the selector has three external inputs and one internal output. Some of the variables are described earlier, however the new variables are described as follow.

- 'id', "(:num)".  
This represents the current block of the state machine, i.e. if  $w=3$  and  $id=1$  then the current next state variable is the first variable in the SISM block.
- 'out', "(: num  $\rightarrow$  time  $\rightarrow$  bool)".  
This function represents all possible outputs for each next state variable under the current control input.

**The MUX Module** The MUX module is a function that takes  $2^w$  inputs and present one value to the output based on the current state. The following predicate describes the behavior of MUX:

```

mux=
 $\vdash_{def}$  mux w input cs out =
    ( $\forall t:time. (out\ t) = (input\ (val\ w\ ((ABS\ w\ cs)\ t))\ t))$ 
);;
```

Referring to the definition and to Figure 3, the MUX module has two external inputs, one internal input, and one internal output. The internal inputs and outpus are described as follows.

- **'input'**, " $(: num \rightarrow time \rightarrow bool)$ ".

This is the data provided by the previous module. It is a bit vector of length  $2^w$ , which represent all possible next state entries.

- **'output'**, " $(: time \rightarrow bool)$ ".

This is the value selected by the current state as one of the next state variable at the next clock pulse.

**The D-ff Module** The D-ff module is a memory device that present the input to the output at the next clock pulse. The predicate that describes the behavior specification is as follows:

```
d-ff=
⊢def d_ff in ld clr q =
    (∀ t:time . q(t+1) = ((clr t) → F |
                        (ld t) → in t | q t))
    ∧ (q 0 = F)"
);;
```

Referring to the definition and to Figure 3, the following variables are defined,

- **'in'**, " $(: time \rightarrow bool)$ ".

This is the next state variable provided by the previous module to be presented to the output at the next clock pulse.

- **'q'**, " $(: time \rightarrow bool)$ ".

This is the output value which constitute one of the variables that when combined with the other outputs from the other blocks, result in the current state.

## 5 Verification

The goal of the verification is stated in logic as follows:

```
"∀ w g data clr ld cs.
    sism_imp w g data clr ld cs ⇒
    sism_spec w g (DATA_ABS w data) clr ld (ABS w cs)"
```

The goal states that the structural implementation implies the behavioral description of the circuit, or, that the behavior follows from the structure. In the goal, DATA-ABS and ABS are two functions used to abstract the signals *w*, *data* and *cs* which are defined at the structural level to behavioral level signals.

The verification is approximately 60% done. The proof is carried out using induction on the width of the SISM. HOL provides mechanical support for induction, rewriting, case analysis and other necessary proof techniques.

## 6 Conclusion

This paper presents the design for a SISM that is being proven to work correctly. This is especially significant because the design of the SISM is very general. Future work will entail tying the structural specification to the actual circuit and using this work to verify specific state machines based on the SISM design.

## References

- [1] Paul Loewenstein, "Reasoning about State Machines in Higher-Order Logic", In M. Leeser and G. Brown, editors, Workshop on Hardware Specification, Verification, and Synthesis, Mathematical Aspects, SpringerVerlag 1989.
- [2] S. Whitaker, G. Maki, and M. Shamanna, "Reliable VLSI Sequential Controllers", NASA Space Engineering research Center, Symposium on VLSI 1990. University of Idaho.
- [3] G. Birtwistle and P. A. Subrahmanyam, Editors. "Current Trends In Hardware Verification And Automated Theorem Proving", Springer-Verlag New York Inc. 1989. pp 4-19.
- [4] M. Alahmad, "Reconfigurable Sequence Invariant State Machine", Masters Thesis. University of Idaho. Dec. 1991.



# Ultra Low Power CMOS Technology

J. Burr and A. Peterson

Space, Telecommunications, and Radioscience Laboratory

Department of Electrical Engineering

Stanford University

Stanford, Ca. 94305

burr@mojave.stanford.edu

**Abstract** - This paper discusses the motivation, opportunities, and problems associated with implementing digital logic at very low voltages, including the challenge of making use of the available real estate in 3D multichip modules, energy requirements of very large neural networks, energy optimization metrics and their impact on system design, modeling problems, circuit design constraints, possible fabrication process modifications to improve performance, and barriers to practical implementation.

## 1 Introduction

As technology continues to scale into the submicron regime, massively parallel architectures are increasingly being constrained by power considerations. Minimizing the energy per operation throughout the system is assuming increasing importance. We are investigating "Ultra Low Power CMOS" to reduce the energy per operation in massively parallel signal processors, microsatellites, and large scale neural networks. We are investigating operating with supply and threshold voltages of a few hundred millivolts to reduce energy per operation by a more than 100 times.

In this paper, we show that minimum energy per operation is achieved in the sub-threshold regime, and that the optimum performance is obtained when  $V_{dd} = V_t$  and  $G_{nd} = V_t - V_{dd}$ . We also show that minimum energy  $\times$  time occurs when  $V_{dd} = 3V_t$ . We show that  $V_t$  should be chosen such that  $I_{on}/I_{off} = ld/a$ , where  $ld$  is the logic depth and  $a$  is the activity ratio, the fraction of gates which are switching at any given time. We also show that  $ld = 11$  minimizes energy in a 32x32 bit parallel multiplier.

## 2 Motivation

The application domains we are targeting include wideband spectrometers requiring  $10^{12}$  operations per second, microsatellites with 100mW power budgets, large scale neural networks requiring  $10^{15}$  connections per second and 1fJ per connection, and small, massively parallel digital signal coprocessors.

As an example, a single SBus slot in a Sun SPARCstation occupies about  $200\text{cm}^3$ , can accommodate over  $2000\text{cm}^2$  of active silicon using 3D stacked multichip module technology, and has a power budget of 10W (see Fig 1). An architecture with a power density of  $2\text{W}/\text{cm}^2$  and 40 MIPS per chip, typical of modern microprocessors, would dissipate 4KW

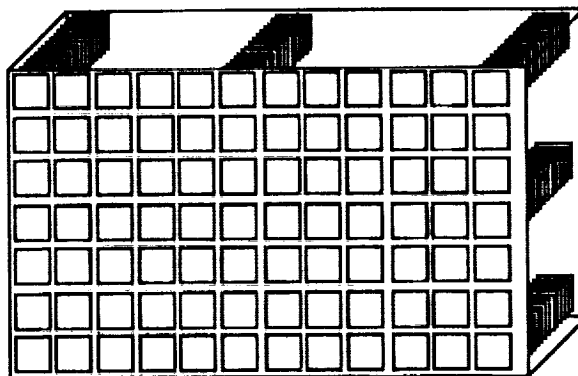


Figure 1: 3D MCM in an SBus slot: 2000 cm<sup>2</sup>, 10W max. V<sub>dd</sub> = 0.7V permits 10 GIPS.

if tiled over the available area and achieve 80 billion operations per second. Only 5 cm<sup>2</sup> of silicon can be used at 10W, yielding 200 MIPS. If the supply voltage is lowered to 700mV, each chip would dissipate 5mW, and the entire 2000cm<sup>2</sup> could be used to achieve 10 billion operations per second at 10W.

### 3 Background

Low voltage digital logic is not new. Richard Swanson described a 100mV CMOS ring oscillator in [6]. Eric Vittoz discussed subthreshold design techniques used in the digital watch industry in [4]. Carver Mead described a variety of subthreshold analog circuits for neural networks in [1]. We believe that low voltage circuits can be used effectively for massively parallel computation in power constrained environments, and that lowering the voltage in submicron technologies has the added benefit of maintaining manageable signal frequencies at the system level.

### 4 Transistor Current

The following equations [6,7] describe drain current as a function of gate voltage, as shown in Fig 2.

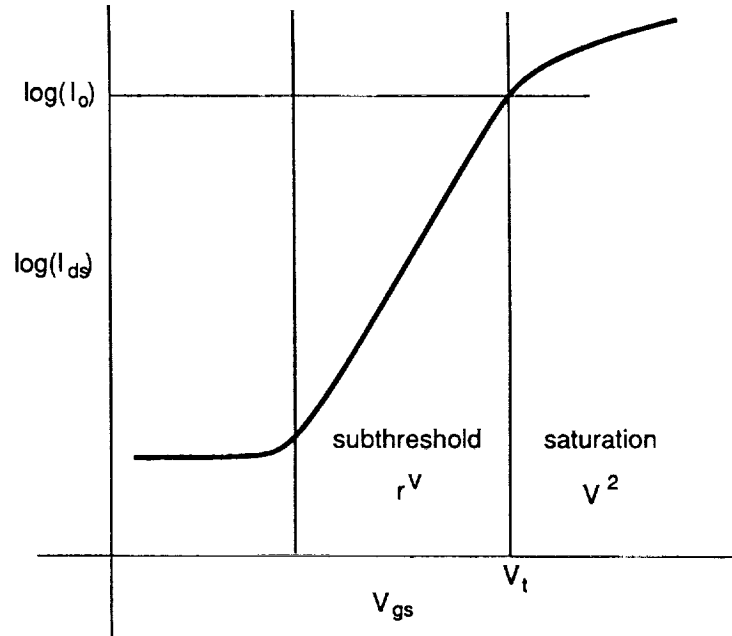


Figure 2: Transistor current vs voltage. Current is exponential with voltage below  $V_t$ , and quadratic above  $V_t$ .

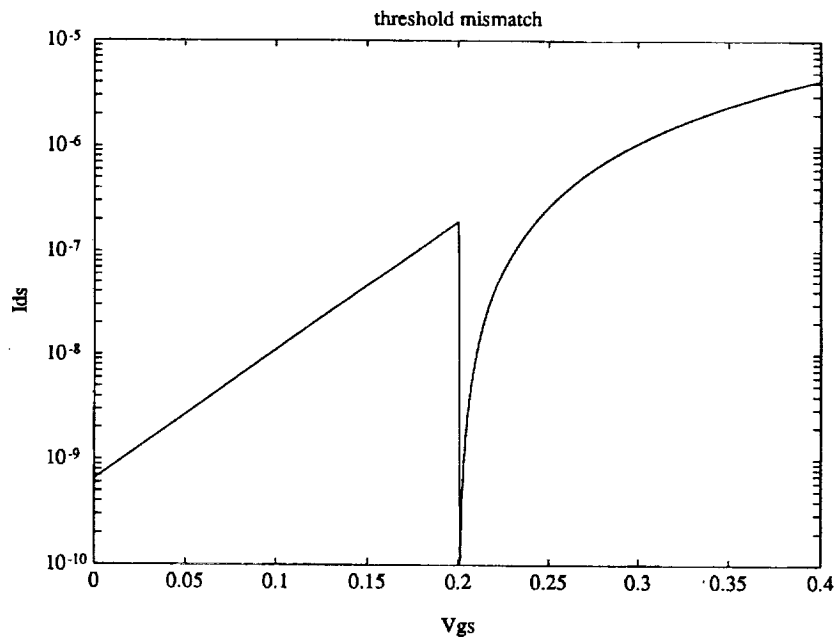


Figure 3: Model discontinuity at  $V_{gs} = V_t$ . The subthreshold model says  $I_{ds} = knV_T^2$ . The saturation model says  $I_{ds} = \frac{k}{2}(V_{gs} - V_t)^2 = 0$ . In the figure  $V_t = 200\text{mV}$ .

subthreshold:  $V_{gs} < V_t$ ;  $I_0 = knV_T^2$   

$$I_{ds} = I_0 e^{\frac{V_{gs}-V_t}{nV_T}} (1 - e^{-\frac{V_{ds}}{V_T}})$$

saturation:  $V_t < V_{gs} < V_{ds} + V_t$   

$$I_{ds} = \frac{k}{2}(V_{gs} - V_t)^2$$

linear:  $V_{ds} + V_t < V_{gs}$   

$$I_{ds} = \frac{k}{2}(2(V_{gs} - V_t)V_{ds} - V_{ds}^2)$$

where  $V_{gs}$  is the gate-source voltage,  $V_t$  is the threshold voltage,  $I_{ds}$  is the drain current,  $k$  is the transconductance in  $A/V^2$ ,  $n$  is the gate coupling coefficient, usually around 0.7,  $V_T$  is the thermal voltage, 0.026V, and  $I_0$  is the current at  $V_{gs} = V_t$ .

Note the exponential dependence of current on voltage below  $V_t$ , and the quadratic dependence above  $V_t$ . These equations do a poor job of modeling behavior in the neighborhood of  $V_t$  (see Fig 3).

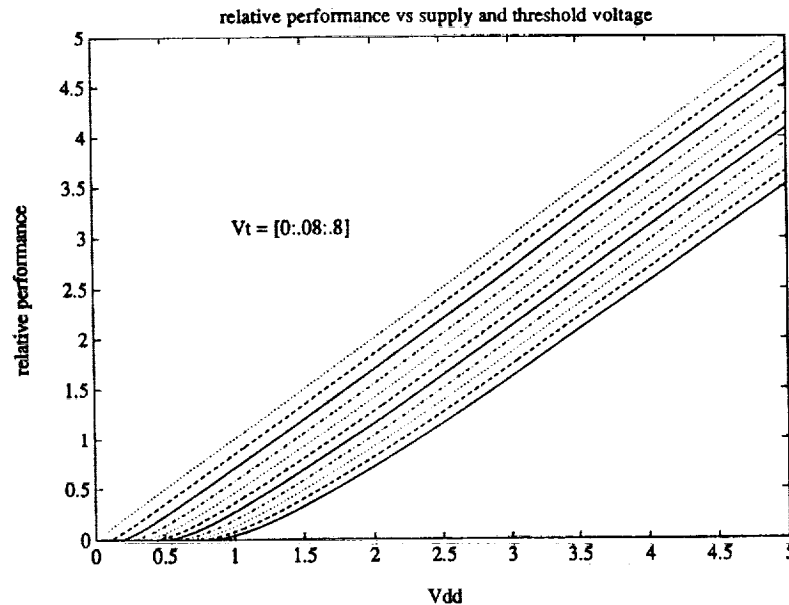


Figure 4: Performance vs voltage for different values of  $V_t$ .

Performance can be approximated when the supply voltage is over threshold by

$$f = I/Q = \frac{k}{2}(V - V_t)^2/(CV).$$

where  $f$  is the clock frequency,  $k$  is transconductance, and  $C$  is the capacitance being switched.

## 5 Optimum Logic Depth



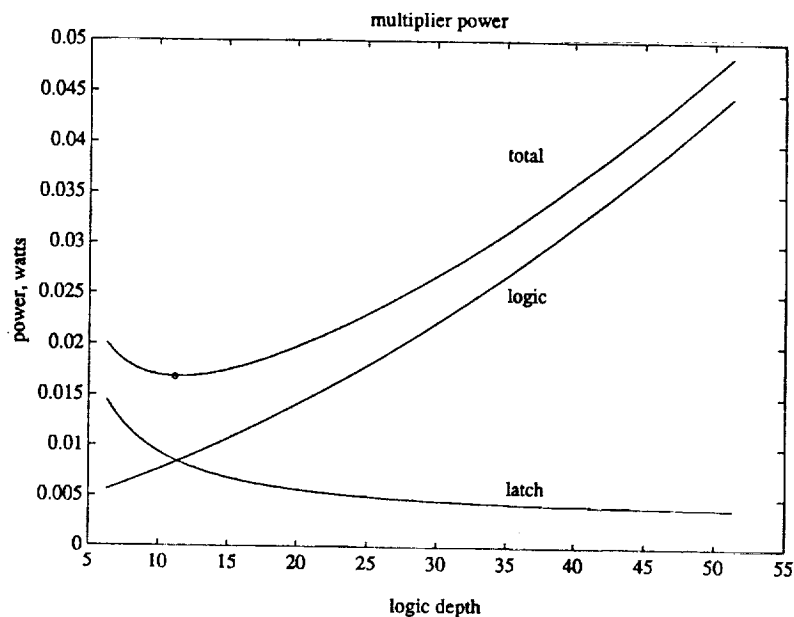


Figure 5: Optimum logic depth of a  $32 \times 32$  bit tree multiplier. For a given  $ld$ , the supply voltage is lowered to match the unpiped throughput. Minimum power consumption occurs at  $ld = 11$ . Latch energy increases as  $ld$  decreases, eventually exceeding logic energy, which decreases as  $ld$  decreases.

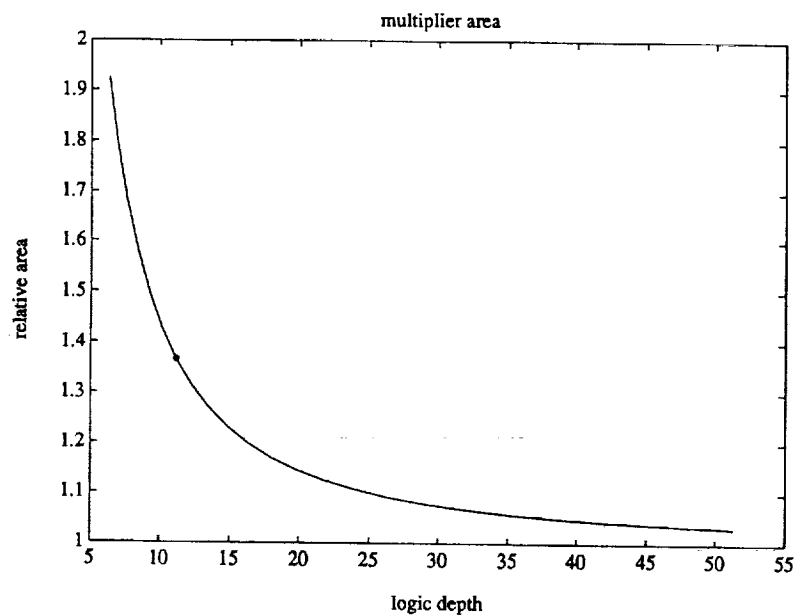


Figure 6: Relative area vs logic depth in a  $32 \times 32$  bit multiplier. The area penalty at  $ld = 11$  is 37%.

We found the optimum logic depth in a  $32 \times 32$  bit tree multiplier by reducing the supply voltage to keep the throughput constant (see Fig 5). We also found the area penalty using this approach (see Fig. 6).  $ld = 11$  is close to the propagation delay through a 4:2 adder [2].

## 6 Minimum Energy

The current available to switch a node is the difference between the current of the ON device and the leakage current of the OFF device. In standard CMOS,  $V_t$  is so high that  $I_{off}$  can be ignored, but in low voltage applications it can be an appreciable fraction of  $I_{on}$ :

$$\begin{aligned} t_{pd} &= \frac{Q}{I} = \frac{C_g V}{I} = \frac{C_g V}{I_{on} - I_{off}} \\ E_{dc} &= I_{off} V l_d t_{pd} = C_g V^2 \frac{l_d}{\frac{I_{on}}{I_{off}} - 1} \\ E_{ac} &= \frac{1}{2} a C_g V^2 \\ E &= E_{ac} + E_{dc} = \frac{1}{2} C_g V^2 \left( a + \frac{2l_d}{\frac{I_{on}}{I_{off}} - 1} \right) \end{aligned}$$

$E$  is minimum when  $I_{on}/I_{off}$  is maximum. Referring to Fig 2,  $I_{on}/I_{off}$  is maximum and constant in the subthreshold region.

In the subthreshold region, if  $V_{ds} = V = V_{hi} - V_{lo}$ , then  $I_{on}/I_{off} = e^{(V_{hi} - V_{lo})/(nV_T)} = e^{V/(nV_T)}$ , so  $E$  depends only on  $V = V_{hi} - V_{lo}$ . Therefore, for a given Vdd, energy is constant in the subthreshold region. For maximum performance at minimum energy, set  $V_{hi} = V_t$  and  $V_{lo} = V_t - V_{dd}$ .

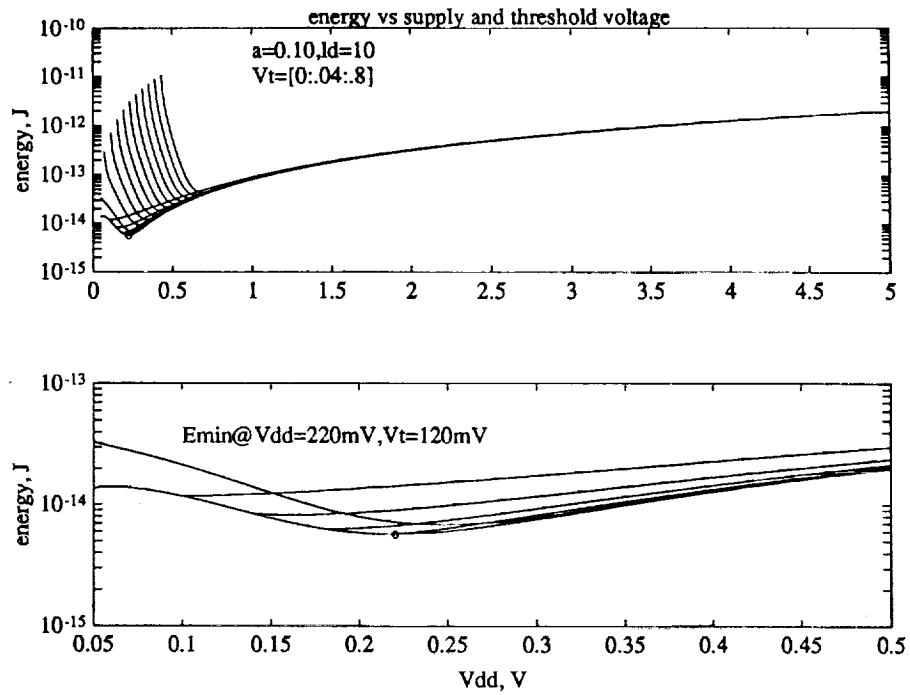
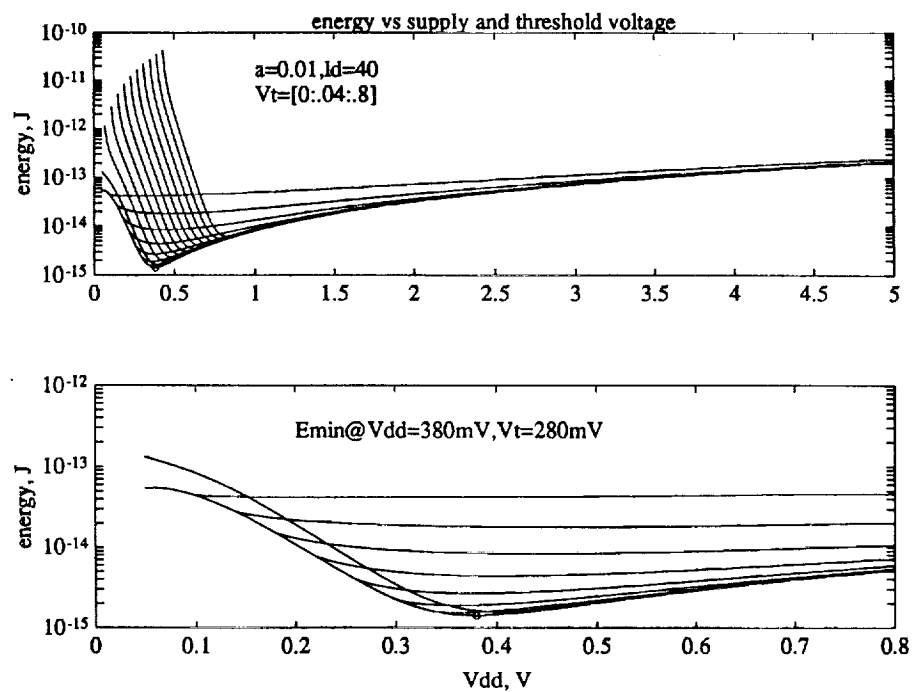
DC energy rises exponentially as Vdd decreases. AC energy rises quadratically as Vdd increases. For optimum  $V_t$ ,

$$\begin{aligned} P_{ac} &= a C V^2 f \\ P_{dc} &= I_{off} V \\ I_{on} &= l_d C V f \end{aligned}$$

If  $P_{ac} = P_{dc}$  and  $V_{dd} = V_t$ , then

$$\begin{aligned} I_{on}/I_{off} &= l_d/a = e^{V_t/(nV_T)} \\ V_t &= nV_T \ln(I_{on}/I_{off}) \end{aligned}$$

Figs 7 and 8 show energy vs Vdd. Table 1 lists the voltages and energies at the global minima.

Figure 7: Energy vs supply voltage for  $a = 0.10$ ,  $ld = 10$  in  $2\mu$  CMOSFigure 8: Energy vs supply voltage for  $a = 0.01$ ,  $ld = 40$  in  $2\mu$  CMOS

## 7 Minimum Energy x Time

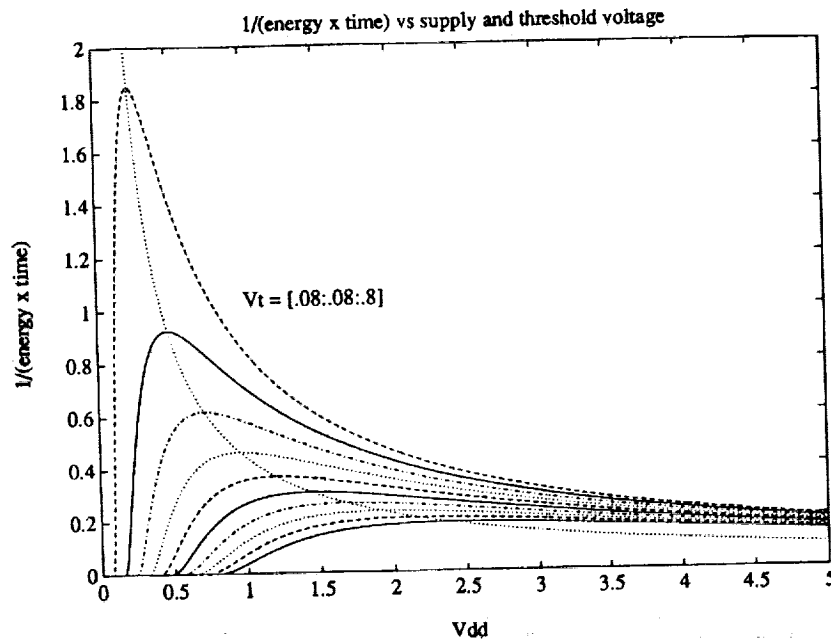


Figure 9:  $1/(\text{energy} \times \text{time})$  vs  $V_{dd}$  and  $V_t$ .  $Et_{\min}$  occurs at  $3V_t$ .

The minimum energy solution is quite slow. Performance should improve dramatically in deep submicron and with low voltage process optimizations. An alternative approach is to minimize energy  $\times$  time. If we assume transistors operate mostly in saturation, then

$$Et = V^2 Q / I = V^3 / (V - V_t)^2$$

$$Et_{\min} = \frac{9}{4} V_t \text{ at } V = 3V_t$$

Fig 9 shows a maximum at  $3V_t$  which grows much more pronounced at low voltage.

## 8 Circuit Design Constraints

A number of interesting circuit design constraints appear when leakage currents are large, and when the dependence of current on voltage is exponential. Three constraints we have observed to date:

- Dynamic circuits are difficult to manage. A minimum size transistor will have a leakage current of about 1nA at  $V_t = 160\text{mV}$ . A dynamic storage node with 100fF of capacitance will hold 50fC of charge at  $V_{dd} = 0.5\text{V}$ . A change of 100mV requires movement of 10fC.  $10\text{fC}/1\text{nA} = 10\text{usec}$ .
- Exponential dependence of current on voltage makes pass transistor logic difficult to use. nfets cannot pass ones and pfets cannot pass zeros. In particular, using nfets as access transistors for static latches does not work.

parameter	negative	positive
reduce $X_j$	increase $R_S, R_D$	decrease $cjsw, cgso, cgdo$
reduce $T_{ox}$	decrease $V_{g,max}$ (gate-src breakdown) increase $C_{ox}$ (increase energy)	increase $k$  decrease $n$
reduce $N_B$	decrease $V_{d,max}$ (punchthrough)	increase $u_o$ decrease $cj, cjsw, n$
reduce $N_G$	increase $R_G$	decrease $V_t$
reduce $N_D$	increase $R_S, R_D$	decrease $cj, cjsw$

Table 2: Process optimization opportunities.

- Fully static logic appears to work well. Transmission gate latches work nicely. SRAM seems to work well, since one of the bitlines will be pulling down on a write.

## 9 Process Optimization

The opportunity exists to improve performance by optimizing fabrication processes for low voltage operation. Carrier mobility degrades significantly in submicron processes as channel doping is increased to prevent punchthrough in the presence of strong electric fields. Reduced voltage operation results in weaker fields, permitting lower channel doping which results in higher carrier mobility and increased transconductance.

Reduced voltage operation also permits lower diffusion doping, since higher diffusion resistance will not impact circuit performance due to reduced transistor drain current. This reduces diffusion capacitance to a negligible fraction of gate capacitance. The only drawback of reducing diffusion doping is that lateral diffusion is reduced, increasing the effective channel length. This is partially offset by the reduced Miller effect since the gate-drain overlap capacitance is reduced. Table 2 summarizes the impact of various process modifications on energy and performance.

While a lower bound of 60mV/decade is achievable at room temperature ( $dV = nV_T \ln(10)$  with  $n = 1$ ),  $dV$  is more typically 80mV/decade in  $2\mu$  CMOS and 90mV/decade in  $0.8\mu$  CMOS.  $T_{ox}/d_0$  can be reduced by reducing  $N_B$ , since  $d_0 = \sqrt{2\epsilon_{si}\phi_{ss}/(qN_B)}$ , where

$\phi_{ss} = V_T \ln(N_B/n_i)$  and  $n_i = \sqrt{1.5T^3 e^{-1.15/V_T}} \times 10^{16}$  [5].

Low gate, drain, and threshold voltages permit all doping concentrations to be reduced, once again due to lower electric field strength. This has two benefits for low voltage operation:

1.  $n$  is reduced, decreasing the subthreshold slope and thus reducing the supply voltage (and therefore energy per operation) necessary to achieve the desired on/off current ratio.
2. source/drain capacitances are reduced, further reducing energy per operation.

## 10 Barriers to Practical Implementation

A number of practical considerations place a lower bound on supply voltage. These are: external interfacing, controlling device thresholds, maintaining adequate noise margins, power supply design, power consumption of OFF devices, and circuit speed. Multichip module packaging provides the opportunity to isolate low-voltage subsystems from other system components. Limits to low voltage operation may be determined to a large extent by the power dissipation in level-shifting interface circuits. Device thresholds have been observed to vary with transistor geometry and even location on a chip [3].

A 10 watt power supply will have to deliver 20amps at  $V_{dd} = 500\text{mV}$ .

## 11 CIS Testchip

In the BiCMOS process at Stanford's Center for Integrated Systems, pfet gates are doped p+ and nfet gates are doped n+. This means that if the channel implant is excluded, both devices have thresholds close to zero volts.  $V_t$  can then be adjusted by adjusting the substrate bias voltage. We have implemented a test chip which contains a number of simple circuit structures (see Fig 10), and will hopefully have some results in time for the conference. The chip has the following characteristics:

- Pfet gates doped p+ have  $V_t \approx 0V$
- Independent substrate and well biases
- self-testing convolutional coder
- ring oscillator
- VCO
- single nfet, pfet, nand, latch



- [2] James B. Burr and Allen M. Peterson, "Energy considerations in multichip-module based multiprocessors ", *IEEE International Symposium on Circuits and Systems*, 1991.
- [3] Aleksandra Pavasovic and Andreas G. Andreou and Charles R. Westgate, " Characterization of CMOS process variations by measuring subthreshold current", *Nondestructive Characterization of Materials IV*, Plenum Press, 1991.
- [4] Eric A. Vittoz, " Micropower techniques", *Design of MOS VLSI Circuits for Telecommunications*, Prentice-Hall, 1985.
- [5] James R. Pfister, " Performance limits of CMOS very large scale integration", PhD thesis, Stanford University, 1984.
- [6] Richard M. Swanson, " Complementary MOS transistors in micropower circuits ", PhD thesis, Stanford University, 1974.
- [7] David A. Hodges and Horace G. Jackson, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill, 1983.



# Parallel Optimization Algorithms and Their Implementation in VLSI Design

G. Lee and J. J. Feeley  
Department of Electrical Engineering  
University of Idaho  
Moscow, ID 83843

**Abstract-** Two new parallel optimization algorithms based on the simplex method are described. They may be executed by a SIMD parallel processor architecture and be implemented in VLSI design. Several VLSI design implementations are introduced. An application example is reported to demonstrate that the algorithms are effective.

## 1 Introduction

Optimal system control is an important part of modern control theory. The kernel problem is optimizing the behavior of systems, as in minimizing the energy or cost required to accurately reach some required terminal state. The search for the control which attains the desired objective while minimizing (or maximizing) a defined system criterion constitutes the fundamental problem of optimal control [1][2][3].

To date, practical applications of optimal control theory are still quite few in number. For a class of systems with fast response, the implementation of a real-time on-line optimal controller has been difficult. The time-consuming computation required for optimal control solutions has been a major obstacle. Modern supercomputers with parallel processing architectures and very fast computation speed are not a practical solution because of their weight, size and cost. Fast computation, small size and low cost are basic requirements for the controller. In this paper, the technique of an algorithmically specialized computer is suggested to achieve an optimal controller which can realize both real-time computation and on-line control for a rapidly responding system. Effective algorithms, parallel architecture, and VLSI implementation are involved in the design of the controller.

Efficient optimization algorithms are very necessary for solving the two-point boundary-value (TPBV) problems which arise in optimal control. Chazan and Miranker in 1970 [4] originally proposed a nongradient-based parallel search algorithm for unconstrained minimization which is suitable for execution using an array of parallel processors. The algorithm involves the parallel execution of  $n$  linear searches along the same direction, starting from  $n$  points, when the dimension of the vector of unknowns is  $n$ . Travassos and Kaufman [5] have applied the algorithm to the solutions of optimal control systems. Housos and Wing in 1984 [6] reported a parallel pseudo-conjugate direction algorithm that performs a set of  $n$  linear searches in parallel along different search directions. Those parallel optimization algorithms proceed by univariate optimization so that they are MIMD-type algorithms [7]. Although they may be used to solve the optimal control problem, it is not easy to shift them to VLSI design for a small size and low cost controller. Two new parallel,

nongradient-based algorithms for unconstrained optimization are presented in this paper. In contrast to existing parallel optimization algorithms, the new parallel algorithms are based on a simplex method and are SIMD-type algorithms [7]. The advantage of the new algorithms is that they do not need a linear search and may be easily shifted to VLSI implementation.

Three kinds of design schemes: digitally controlled analog, hybrid, and pure digital, are presented in this paper. Their VLSI implementation and their performances are discussed.

## 2 New parallel optimization algorithms

The following unconstrained minimization problem is considered:

$$\min f(X), X \in \mathbb{R}^n,$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , and is usually non-quadratic and nonlinear.

We wish to find a point  $X^*$  numerically such that, if  $\varepsilon > 0$ , then

$$f(X^*) < f(X), \text{ for all } X : \|X - X^*\| < \varepsilon.$$

Two parallel simplex algorithms, PS1 and PS2, which are based on an improved simplex method [8][9] and use parallel function evaluations, are stated below.

**Algorithm PS1:** The algorithm PS1 predicts four candidate vertices simultaneously in one iteration. Therefore at least four parallel processors are required. Each iteration includes two phases: the first is for parallel evaluations and the second is for choosing a new vertex to generate a new simplex via function value comparison. The computation time for the function evaluations is always longer than the time for the function value comparison. The execution of parallel function evaluations effectively reduces computation time since it is a major part of the time for one iteration cycle. It is also important that the parallel function evaluations are of the SIMD type. This allows the algorithm to proceed in the SIMD parallel architecture. The number of parallel function evaluations required by PS1 is only about half the number required by the improved simplex algorithm of Nelder and Mead [8].

The algorithm PS1 is described below:

(0) Initial simplex:

(0a) Set the iteration number  $k = 0$ .

(0b) Starting point  $v^0 = (x_1^0, x_2^0, \dots, x_n^0)$  is given. An initial simplex  $V^0 = [v_1^0, v_2^0, \dots, v_{n+1}^0]$  is formed in parallel by:  $v_1^0 = (1 - \delta)v^0$

$$v_{i+1}^0 = \begin{cases} v^0 + \delta E_i x_i^0, & \text{if } x_i^0 \neq 0 \\ v^0 + \delta E_i, & \text{otherwise} \end{cases}$$

where  $E_i = \{0 \dots 0 \ 1 \ 0 \dots 0\}$ ,  $i = 1, 2, \dots, n$ , and  $\delta = 0.1$

(0c) Parallel evaluation of the function value at the vertices of  $V_0$   
 $Y^0 = [f(v_1^0), f(v_2^0), \dots, f(v_{n+1}^0)]$ .

(1) Parallel sorting:

Set  $k = k + 1$ . Let  $S^k = [X_1^k, X_2^k, \dots, X_{n+1}^k]$  and  $F^k = [f_1^k, f_2^k, \dots, f_{n+1}^k]$  be ordered  $V^{k-1}$  and  $Y^{k-1}$ .

Find  $d, d = \max(\|X_i^k - X_1^k\|), i = 2, 3, \dots, n+1$ , if  $d$  is small enough, then stop, otherwise continue as follows.

Denote

$X_l$  by  $X_1^k$ ,  $X_{sh}$  by  $X_n^k$ ,  $X_h$  by  $X_{n+1}^k$ ,

$f_l$  by  $f_1^k$ ,  $f_{sh}$  by  $f_n^k$ ,  $f_h$  by  $f_{n+1}^k$ .

The centroid  $\bar{X}$  is the mean of the vertices with  $i \neq n+1$ , i.e.,

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i^k$$

(2) Parallel computation:

$$X_c = (1 - \beta)\bar{X} + \beta X_h$$

$$X_a = (1 + \beta)\bar{X} - \beta X_h$$

$$X_r = (1 + \alpha)\bar{X} - \alpha X_h$$

$$X_e = (1 + \gamma)\bar{X} - \gamma X_h$$

Parallel function evaluations

$$f_c = f(X_c), f_a = f(X_a), f_r = f(X_r) \text{ and } f_e = f(X_e)$$

(3) Comparison and selection of new point for updating simplex:

(3a) If  $f_e < f_r < f_l$ , then  $X_h = X_e$  and  $f_h = f_e$ .

(3b) If  $f_{sh} > f_r > f_l$  or  $f_e > f_r > f_l$ , then  $X_h = X_r$  and  $f_h = f_r$ .

(3c) If  $f_h > f_r > f_{sh}$  and  $f_a < f_{sh}$ , then  $X_h = X_a$  and  $f_h = f_a$ .

(3d) If  $f_r > f_h$  and  $f_c < f_{sh}$ , then  $X_h = X_c$  and  $f_h = f_c$ .

(3e) If  $f_r > f_h$  and  $f_c > f_{sh}$  or if  $f_h > f_r > f_{sh}$  and  $f_a > f_{sh}$ , do shrinkage in parallel:

$X_s = [X_1, X_{s_j}]$ , where  $X_{s_j} = (X_j + X_1)/2, j = 2, 3, \dots, n+1$ , evaluate and update  $F^k = [f_1, f(X_{s_2}), f(X_{s_3}), \dots, f(X_{s_{n+1}})]$  and  $S^k = X_s$ , then do

(4) Update the simplex:

let  $V^k = S^k, Y^k = F^k$ , then return to (1).

**Algorithm PS2:** The algorithm PS2 is developed from the algorithm PS1 by increasing the parallel processors to sixteen. Twenty processors in total are utilized to predict twenty candidate vertices simultaneously in one iteration. The algorithm PS2 is more effective than the algorithm PS1. One iteration of the algorithm PS2 is functionally equivalent to two iterations of the algorithm PS1. Thus the algorithm PS2 will do the same function in roughly half the time of the algorithm PS1. Algorithm PS2 is also of the SIMD-type.

The algorithm PS2 is described below:

(0) The same as Step (0) of Algorithm PS1;

(1) The same as Step (1) of Algorithm PS1;

(2) Parallel computation:

(2a) Compute the first level direction points (four in total) in parallel:

$$X_c = (1 - \beta)\bar{X} + \beta X_h$$

$$X_a = (1 + \beta)\bar{X} - \beta X_h$$

$$X_r = (1 + \alpha)\bar{X} - \alpha X_h$$

$$X_e = (1 + \gamma)\bar{X} - \gamma X_h$$

and find 4 conductive points in parallel

$$\bar{X}_i = \frac{1}{n}(\sum_{j=1}^{n-1} X_j + X_i), i = 'c', 'a', 'r', 'e',$$

(2b) Compute the second level direction points (sixteen in total) in parallel:

$$X_{ic} = (1 - \beta)\bar{X}_i + \beta X_{sh}$$

$$X_{ia} = (1 + \beta)\bar{X}_i - \beta X_{sh}$$

$$X_{ir} = (1 + \alpha)\bar{X}_i - \alpha X_{sh}$$

$$X_{ie} = (1 + \gamma)\bar{X}_i - \gamma X_{sh}$$

where  $i = 'c', 'a', 'r', 'e',$

(2c) Parallel function evaluations

$$f_i = f(X_i)$$

$$f_{ic} = f(X_{ic}), f_{ia} = f(X_{ia}), f_{ir} = f(X_{ir}) \text{ and } f_{ie} = f(X_{ie})$$

where  $i = 'c', 'a', 'r', 'e',$

(3) Comparison and updating simplex:

Set  $m = 0$ .

(3a) If  $f_e < f_r < f_l$ ,  $j = 'e',$  or

if  $f_{sh} > f_r > f_l$  or  $f_e > f_r < f_p$ ,  $j = 'r',$  or

if  $f_h > f_r > f_{sh}$  or  $f_a < f_{sh}$ ,  $j = 'a',$  or

if  $f_r > f_h$  and  $f_c < f_{sh}$ ,  $j = 'c',$  set  $m = m + 1$  and do (3b)

if  $f_r > f_h$  and  $f_c > f_{sh}$ , or if  $f_h > f_r > f_{sh}$  and  $f_a > f_{sh}$ , do shrinkage in parallel,  $X_s = \{X_1, X_{s_j}\}$ , where  $X_{s_j} = (X_j + X_1)/2$ ,  $j = 2, 3, \dots, n+1$ , evaluate

$F_k = \{f_1, f(X_{s_2}), f(X_{s_3}), \dots, f(X_{s_{n+1}})\}$  and let  $s^k = X_s$ , then do (4).

(3b) Replacing:

$$X_h = X_{sh}, f_h = f_{sh}, X_{sh} = X_j, f_{sh} = f_j, \text{ and } X_i = X_{ji}, f_i = f_{ji}.$$

If  $m = 1$  do (3a), otherwise do (4).

(4) The same as the step (4) of the algorithm PS1.

### 3 Example of application to real-time optimal control

The air-to-air missile-target intercept is a practical real-time optimal control application. A typical intercept mission from missile launch to intercept, may take only a few seconds. It is almost impossible to achieve true optimal control during such a short time interval with present technology. The efficiency of the algorithm PS2 for real-time application of optimal control is demonstrated in this section via simulation of a 3-dimensional air-to-air missile-target intercept problem. An optimal guidance law that minimizes missile energy expenditure with fixed final time  $t_f$  and fixed final state (zero miss range) is derived in Ref [9] using nonlinear optimal control theory. This section focuses on solving the nonlinear TPBV problem (NTPBVP) which arises in the intercept problem by the "shooting method" using Algorithm PS2.

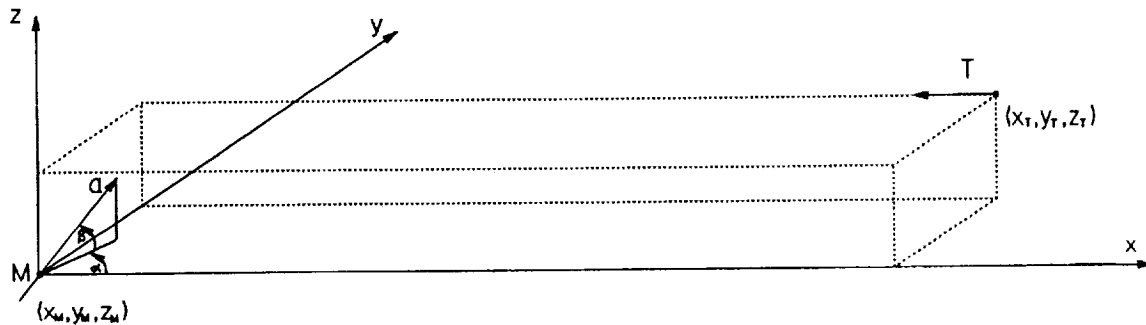


Figure 1: 3-dimensional intercept geometry

Figure 1 shows the 3-dimensional intercept scenario. The target **T** moves in a straight line at constant velocity  $v_T$  and the missile **M** moves at controlled acceleration  $a(t)$  and its direction angles are  $\alpha(t)$  and  $\beta(t)$ . An on-board optimal controller in the missile calculates and provides,  $a(t)$ ,  $\alpha(t)$  and  $\beta(t)$  to the missile thruster.

The NTPBVP obtained is

$$\begin{aligned}
\dot{x}_1 &= x_4 & x_1(0) &= x_{10}, & x_1(t_f) &= 0 \\
\dot{x}_2 &= x_5 & x_2(0) &= x_{20}, & x_2(t_f) &= 0 \\
\dot{x}_3 &= x_6 & x_3(0) &= x_{30}, & x_3(t_f) &= 0 \\
\dot{x}_4 &= -a(x_{10}^2 + x_{11}^2)x_{10} & x_4(0) &= x_{40} \\
\dot{x}_5 &= -a(x_{10}^2 + x_{11}^2)x_{11} & x_5(0) &= x_{50} \\
\dot{x}_6 &= -ax_{12} & x_6(0) &= x_{60} \\
\dot{x}_7 &= 0 \\
\dot{x}_8 &= 0 \\
\dot{x}_9 &= 0 \\
\dot{x}_{10} &= x_7 & x_{10}(t_f) &= 0 \\
\dot{x}_{11} &= x_8 & x_{11}(t_f) &= 0 \\
\dot{x}_{12} &= x_9 & x_{12}(t_f) &= 0
\end{aligned} \tag{1}$$

where

$$a = (x_{10}^2 + x_{11}^2)^2 + x_{12}^2.$$

Notice that the initial conditions  $x_{10}$  to  $x_{60}$  are constant and the terminal values  $x_1(t_f)$  to  $x_3(t_f)$  and  $x_{10}(t_f)$  to  $x_{12}(t_f)$  are zeros. The first six equations of (1) are the dynamics of the system. The second six equations are the co-state equations. The shooting method starts with estimating a set of initial values  $(x_7(0)x_8(0)x_9(0))^T$ , then integrates (1) forward, with given and estimated initial values  $x_1(0)$  to  $x_{12}(0)$ . The resulting terminal values are usually different from the given ones. An error function  $E$  is defined by

$$E = \sqrt{x_1(t_f)^2 + x_2(t_f)^2 + x_3(t_f)^2 + x_{10}(t_f)^2 + x_{11}(t_f)^2 + x_{12}(t_f)^2} \tag{2}$$

The shooting method attempts to minimize the error function  $E$ :

$$\min E \rightarrow 0 \tag{3}$$

This can be done by means of the algorithm PS2 to update the estimated initial values until (3) is satisfied.

The initially given condition is <sup>1</sup>

$$\begin{aligned}
x_{10} &= 20000 \text{ (ft)} \\
x_{20} &= 3000 \text{ (ft)} \\
x_{30} &= 2500 \text{ (ft)} \\
x_{40} &= -972 \text{ (ft/sec)} \\
x_{50} &= -972 \text{ (ft/sec)} \\
x_{60} &= 0 \text{ (ft/sec)}
\end{aligned}$$

and the fixed final time is  $t_f = 5(\text{sec})$ .

Assume the target velocity  $v_T$  is constant, the travel path of the target will be a straight line. The target path may be calculated correctly by

<sup>1</sup>Data taken from Ref. [10]

$$\begin{aligned}x_T(t) &= x_{0T} + v_T t \\y_T(t) &= y_{0T} \\z_T(t) &= z_{0T}\end{aligned}$$

where  $[x_{0T} \ y_{0T} \ z_{0T}]^T$  is the target's initial position so that open-loop optimal control may be employed by the missile.

To come up with open-loop optimal control numerically, one must first solve the NTP-BVP (1). For a set of rough initial estimations

$$\begin{aligned}x_{70} &= 1 \\x_{80} &= 1 \\x_{90} &= 1\end{aligned}$$

using the algorithm PS2, the resulting solutions are in Table 1:

TI (sec)	OIV	PFE	CMR (ft)	RMR (ft)
[0 5]	$x_7(0)=0.65993$ $x_8(0)=-0.08107$ $x_9(0)=0.92175$	114	0	1.287e-9

Table 1: The numerical results of the intercept scenario

TI—Time interval,  
OIV—Optimal initial values,  
PFE—Parallel function evaluations,  
CMR—Constraint of miss range,  
RMR—Real miss range.

As a rough estimation of computation time, if the PFE < 120 in five seconds as shown in Table 1, then the real-time optimal control can be implemented for this air-to-air missile-target intercept problem. This is very possible with modern VLSI techniques. In the next section several VLSI design possibilities are introduced.

## 4 VLSI implementations

This section presents design possibilities for potential real-time, on-line, optimal controllers. These optimal controllers will be algorithmically specialized parallel computers





demultiplexer (DMul) and a set of updating switches (USw), are saved in the simplex memories (SMe). The Mul and the DMul also pass the initial simplex vertexes, denoted as  $[X_1 \cdots X_{n+1}]$ , to the SMe. Then a basic simplex with its function values is stored for further operations.

According to the algorithm PS1, the stored simplex must be updated. To do this, the simplex in the SMe must be first sorted by a sorting circuit (Sorting). A sorted simplex,  $[X_l, \cdots, X_{sh}, X_h]$  with function values  $[f_l, \cdots, f_{sh}, f_h]$ , is available at the output of the Sorting. From it four direction points,  $X_c$ ,  $X_a$ ,  $X_r$  and  $X_e$ , can be found in the direction points module (DP). Similar to the initial simplex, they and their function values,  $f_c$ ,  $f_a$ ,  $f_r$  and  $f_e$ , evaluated by the PFE are stored in the direction memories (DMe) via the Mul and the DMul. A new point module (NP) compares  $[f_c, f_a, f_r$  and  $f_e]$  with  $[f_l, f_{sh}$  and  $f_h]$  and then selects a proper one of the direction points, denoted by  $X'_h$ , with its function value  $f'_h$ . Via the USw the  $X'_h$  replaces the vertex  $X_h$  to update the basic simplex. The positions of  $X_h$  and  $f_h$  are indexed by one of the signals  $g_1$  to  $g_{n+1}$  generated by the Sorting.

In case no new point can be selected, the NP will send out a digital signal Ds. Through it the DTC generates another control signal Cs to the Mul and the DMul, then a shrunken simplex from the simplex shrinkage (SSh),  $[X_1, X_{s1}, \cdots, X_{sn}]$ , with its function values is passed to the SMe, so that the basic simplex is updated.

The simplex size module (SSi) and the convergence testing module (CT) monitor the size of the sorted simplex and its minimal function value. Together with the size switches (SSw) and the Sorting, when one of them satisfies a given criterion, the CT will send a "stop" signal to finish the iterations.

The digital timing controller (DTC) is necessary to control the timing of the whole system. The functions of the DTC may be stated by defining its inputs and outputs as follows:

#### Inputs:

- Start: actuates the DTC and starts the computation,
- T: a parameter given for setting up the width of the Ce's active interval,
- Repeat: after the computation, reactuates the DCT and repeats the solutions if necessary,
- Stop: stops the iteration when the solutions are available,
- Ds: active when shrinkage simplex is needed, sets up the Cs,

#### Outputs:

- Ce: actuates the PFE, the Mul and the DMul, its active length is given by the input T,
- Ci: passes the initial simplex and its function values to the SMe via the Mul and the DMul,
- Cs: passes the shrinkage simplex and its function values to the SMe via the Mul and the DMul, it is controlled by the input Ds,
- Cd: passes the direction points and their function values to the DMe via the Mul and the DMul,
- Cr: repeats the computation, it is controlled by the input "Repeat",
- Cm: actuates the SMe and the DMe,
- Ct: tests the simplex size, active at each iteration.

The design of the DTC is a normal digital logic design and is not included here.

To meet various application requirements, three kinds of design schemes (1) digitally controlled analog, (2) hybrid, and (3) all-digital, are suggested here. The design of digitally controlled analog is due to analog computation on both the PFE and the main algorithm part. The hybrid design uses digital computation for the main algorithm. Finally the digital design is a pure digital scheme. Due to their different characteristics, they are employed in different circumstances as listed below.

For the digitally controlled analog controller:

1. Accuracy limited but faster computation.
2. Limited memory period.
3. More efficient for low frequency systems with shorter operation time.

For the hybrid controller:

1. Same as 1 above.
2. Unlimited memory period.
3. More efficient for low frequency systems with longer operation time.

For the digital controller:

1. Accuracy unlimited but slow computation.
2. Unlimited memory period.
3. No strong relation to frequency and time of system operation.

## 4.2 Digitally controlled analog scheme

In general, analog computation is faster than digital computation. This suggests the PFE and the main algorithm part (not including the DTC) may be implemented by analog techniques. However analog long-time memory is not easily implemented on a VLSI chip. Memory time is strongly depended on the problem's complexity. If the requirement for memory time is too long and the size requirement is critical, the hybrid computation scheme should be considered as below.

## 4.3 Hybrid scheme

The hybrid scheme includes digital computation and memories in the main algorithm part. But the PFE still uses analog techniques. In practice, the PFE is a parallel electronic differential analyzer (EDA) which consists of some integrators. Integration computations is more convenient with analog circuiting than with digital. Keeping the PFE in analog will reduce computation time. A hybrid scheme is suggested in Figure 3. The system has three sections: the PFE, the digital algorithm processor and the linkage system, in which some analog/digital (A/D) and digital/analog (D/A) converters are essential.

Each numerical value in the digitally controlled analog scheme mentioned above, such as each function value, each element value of a simplex vertex and each constant value, will be described in m-bit form and be stored in a m-bit register. In order to achieve parallel computation, the input and output to these registers are parallel m-bit data buses. Furthermore, all of the digital devices in this system are parallel.

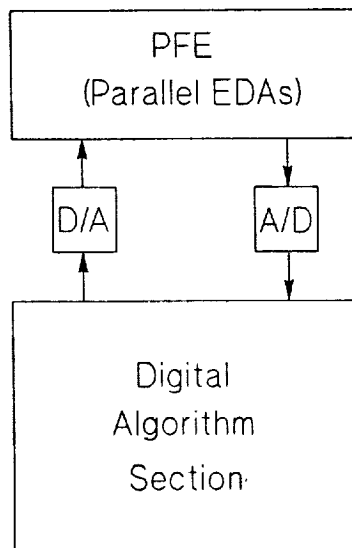


Figure 3: Hybrid scheme

#### 4.4 Digital scheme

Based on the hybrid scheme, a pure digital optimal controller may be obtained by designing a digital PFE. A key point is to design, for the PFE, a digital integrator, which is very different from an analog one. The design of the digital PFE is related to both the solution methods and the particular problem, and may be separated into two parts. The first one is an algorithmically specialized unit of a solving algorithm, such as the Runge-Kutta algorithm, and another is a computing unit of a given differential equation.

### 5 Conclusion

Two new parallel optimization algorithms, PS1 and PS2, based on the simplex method are described. Four processors are required for PS1 and twenty for PS2. They may be executed by a SIMD parallel processor architecture and may be easily shifted to VLSI design.

The numerical result of a 3-dimensional air-to-air missile-target intercept problem has been reported to demonstrate that the algorithms are effective and the real-time optimal controllers are feasible for a class of optimal control systems with fast response.

As a design example, the algorithm PS1 has been shifted to a VLSI implementations. Three types of controller design schemes have been presented: (1) digitally controlled analog, (2) hybrid, and (3) pure digital controller. They can be employed satisfactorily for different application requirements.

In general, the optimal controllers converge rather rapidly, once the estimation of an initial value is found such that the evaluation of the error function  $E$  being minimized results in a number in the neighborhood of zero. However, if the problem to be solved is very sensitive to small perturbations in the initial co-state vector, convergence to an optimal

solution may be slow, or even fail. This case was not considered in this research. To overcome this problem a method [5] suggested by R. Travassos and H. Kaufman may be added in the design of the optimal controllers. This approach is currently under consideration.

## References

- [1] F. L. Lewis, Optimal Control, New York: Wiley, 1986.
- [2] Andrew P. Sage and Chelsea C. White, III, Optimum Systems Control, Second edition, Prentice-hall., Englewood Cliffs, New Jersey, 1977.
- [3] A. E. Bryson, Jr. and Yu-Chi Ho, Applied Optimal Control, Blaisdell Publishing Company, 1969.
- [4] D. Chazan and W. L. Miranker, A Nongradient and Parallel Algorithm for Unconstrained Minimization, *SIAM J. Control*, Vol.8, No.2, pp.207-217, May 1970.
- [5] R. Travassos and H. Kaufman, Parallel Algorithms for Solving Nonlinear Two-Point Boundary-Value Problems Which Arise in Optimal Control, *Journal of Optimization Theory and Applications*, Vol.30, No.1, pp.53-71, Jan. 1980.
- [6] E. C. Housos and O. Wing, Pseudo-Conjugate Directions for the Solution of the Nonlinear Unconstrained Optimization Problem on a Parallel Computer, *Journal of Optimization*, Vol.42, No.2, pp.169-180, Feb. 1984.
- [7] S. Lakshmivarahan and Sudarshan K. Dhall, Analysis and Design of Parallel Algorithms: Arithmetic and Matrix problems, McGraw-Hill, Inc., 1990.
- [8] J. A. Nelder and R. Mead, A Simplex Method for Function Minimization, *Computer Journal*, Vol. 7, pp.308-313, 1965.
- [9] G. Lee, Parallel Computation for Optimal Control Systems, University of Idaho, Ph.D. Dissertation, 1991.
- [10] G. M. Anderson, Comparison of Optimal Control and Differential Game Intercept Missile Guidance Laws, *J. Guidance and Control*, Vol.4, No.2, pp.109- 115, March-April 1981.

## Canonical Multi-Valued Input Reed-Muller Trees and Forms

M. A. Perkowski<sup>1</sup> and P. D. Johnson

Department of Electrical Engineering

Portland State University

P.O. Box 751, Portland, Oregon 97207

**Abstract** - There is recently an increased interest in logic synthesis using EXOR gates. The paper introduces the fundamental concept of *Orthogonal Expansion*, which generalizes the ring form of the Shannon expansion to the logic with multiple-valued (mv) inputs. Based on this concept we are able to define a family of canonical tree circuits. Such circuits can be considered for binary and multiple-valued input cases. They can be multi-level (trees and DAGs) or flattened to two-level AND-EXOR circuits. Input decoders similar to those used in Sum of Products (SOP) PLAs are used in realizations of multiple-valued input functions. In the case of the binary logic the family of flattened AND-EXOR circuits includes several forms discussed by Davio and Green. For the case of the logic with multiple-valued inputs, the family of the flattened mv AND-EXOR circuits includes three expansions known from literature and two new expansions.

### 1 Introduction

Although the EXOR gate exists in most VLSI cell libraries, there are no logic synthesis systems that find optimized multi-level circuits using EXORs. The recently developed PLD devices, such as Programmable Gate Arrays (Xilinx LCA 3000) [33], Signetics LHS501 [32], Actel [7] or other [13], either include EXOR gates, or allow to realize them in the "universal logic modules". Since the five input EXOR gate in Xilinx device has the same speed and cost as, for instance, a five input OR gate [5], the new design methods are needed for such technologies that will assume the usage of EXOR gates on the same full rights as the AND and OR gates. Particularly, if a Reed-Muller [15,22] form has less terms than a two-level AND-OR expression, this form should be used for Xilinx realization, and not the SOP expression, as it is done nowadays.

The problem of finding the minimal generalized Reed-Muller (GRM) canonical form of optimal polarity [14] (called also fixed-polarity Reed-Muller [9]), as well as the problem of finding the minimal Exclusive Sum of Products (ESOP) of a Boolean function [2,10,27,28], are the classical ones in logic synthesis theory, but exact solutions to them have been proposed for only small functions [16,17].

Solving the above two problems, and creating other new methods of multi-level EXOR circuits design is practically important for several reasons: (1) It has long been the experience of logic designers, that the EXOR circuits can be more economical than the

---

<sup>1</sup>This research was supported in part by the NSF Research Initiation Award for the first author

conventional inclusive (AND-OR) normal circuits. This was also confirmed practically on many practical examples, especially on arithmetic and telecommunication circuits [1,12]. It was also proven theoretically [26] on worst case and arithmetic functions. (2) The structure of EXOR circuits implementations is especially suitable for VLSI, optical, and some other recent technologies. The RM and GRM forms have absolutely superior design-for-test properties [6,11,21,23], unmatched by other realizations. This was not used in the past since the EXOR gate realizations were slow and area-expensive. With the arrival of PGA devices this deficiency no longer holds and the theories developed for instance in [6,11,21,23] should be practically used. (3) Currently, the widely used logic minimization programs such as Espresso [3] and MIS II do not take into account EXOR gates in their minimization processes which often causes nonminimal results. There is a growing industrial interest among CAD logic synthesis tools users community to have a program that would generate optimized circuits including EXOR gates [12], and such tools start to be introduced to CAE market (for instance by Mentor Graphics Inc.). (4) The new tools for ESOP synthesis are either heuristic [2,10,18,27,28] or produce exact solutions for general ESOPs [16,17,20], but are so slow that can be applied only to small functions. For few canonical forms included in ESOPs optimal programs exist for functions of about 10 variables [29,30,31]. It is therefore important to construct programs that will be faster than the current exact minimizers and still be able to produce quasi-minimal solutions.

A book by Davio [4] and a paper by Green [9] give information on the numbers and properties of various canonical forms being specializations of binary ESOPs, which may be useful to create efficient algorithms for them. In [19] we presented a family of *multiple-valued input expansions*. In this paper we will present a subset of the family from [19], but we will present the material in a more complete and systematic way. We will introduce new canonical binary and multiple-valued forms and expressions. Forms, Directed Acyclic Graphs (DAGs), Trees and expressions obtained by the introduced here tree searching methods will be all called *expansions*. The ultimate goal of the research reported here is to create synthesis programs, exact and approximate, for all known and some new forms being subsets of binary and multiple-valued input ESOPs.

## 2 Binary Generalizations of Reed-Muller Forms

A *Reed-Muller (RM) expression* (for binary logic) is an exclusive sum of products of *positive* (non-complemented) input variables. A *Negative Reed-Muller (NRM) expression* is an exclusive sum of products of *negative* (complemented) input variables. Both these expansions are called *Single Polarity Reed-Muller Forms*.

**Definition 2.1.** The *literal*  $x_i^c$  is a variable  $x_i$  in either positive (  $x_i$  ) or complemented (  $\bar{x}_i$  ) form.

Let us consider the following form:

$$f(x_1, \dots, x_n) = g_0 \oplus g_1 x_1^c \oplus \dots \oplus g_n x_n^c \oplus g_{n+1} x_1^c x_2^c \oplus \dots \oplus g_{2^n-1} x_1^c x_2^c \dots x_n^c \quad (1)$$

where:  $g_i = 0$  or  $1$ , and  $x_i^c = x_i$  or  $\bar{x}_i$ .

**Definition 2.2.** By a *Generalized Reed-Muller Form (GRM)* one understands a form 1 in which each variable can be complemented (negative) or not complemented (positive), but can not stand in both forms.

Such forms are canonical, which means that only one such form exists for every polarity of variables (there are  $2^n$  such polarities for a Boolean function of  $n$  binary inputs, which means that there are  $2^n$  corresponding GRM forms). Applying the principle of duality to all presented forms one gets the dual forms: the system  $(\oplus, \cdot)$  is replaced with the dual system  $(\odot, +)$ . All results of this paper, after applying the principle of duality to them, hold in the dual system as well. Let us observe that the circuits generated for both systems can be implemented using EXOR and NOR gates or EXOR and NAND gates.

By "flattening" we understand applying the Boolean rule,  $a(b \oplus c) = ab \oplus ac$ . Flattening is used to convert trees and multi-level expressions to two-level expressions, such as Reed-Muller forms, or ESOPs.

The well-known Shannon expansion for the case of ESOP expansion is as follows:

$$f(x_1, \dots, x_i, \dots, x_n) =$$

$$\bar{x}_i \cdot f(x_1, \dots, x_i = 0, \dots, x_n) \oplus x_i \cdot f(x_1, \dots, x_i = 1, \dots, x_n) \quad (2)$$

By applying laws  $\bar{a} = 1 \oplus a$  and  $a = 1 \oplus \bar{a}$  one gets:

$$f(x_1, \dots, x_i, \dots, x_n) =$$

$$f(x_1, \dots, x_i = 0, \dots, x_n) \oplus x_i \cdot [f(x_1, \dots, x_i = 0, \dots, x_n) \oplus f(x_1, \dots, x_i = 1, \dots, x_n)] \quad (3)$$

and

$$f(x_1, \dots, x_i, \dots, x_n) =$$

$$f(x_1, \dots, x_i = 1, \dots, x_n) \oplus \bar{x}_i \cdot [f(x_1, \dots, x_i = 0, \dots, x_n) \oplus f(x_1, \dots, x_i = 1, \dots, x_n)] \quad (4)$$

In the short form:

$$f = x_i \cdot f_{x_i} \oplus \bar{x}_i \cdot f_{\bar{x}_i} \quad (5)$$

$$f = f_{\bar{x}_i} \oplus x_i \cdot [f_{x_i} \oplus f_{\bar{x}_i}] \quad (6)$$

$$f = f_{x_i} \oplus \bar{x}_i \cdot [f_{x_i} \oplus f_{\bar{x}_i}] \quad (7)$$

Let us observe that these expansion formulas have been applied by several authors for the synthesis of GRM forms for completely specified functions [4]. Davio [4] and Green [9] use them as a base of *Kronecker Reed-Muller (KRM)*, *Pseudo-Kronecker Reed-Muller (PKRM)*, and *Quasi-Kronecker Reed-Muller (QKRM)* forms (Green uses also trees for better explanation). If only rule 6 is used repeatedly for some fixed order of expansion variables, the RM Trees are created, which correspond to RM forms after their flattening. If for every variable one uses either rule 6 or rule 7, the GRM Trees are created, from which GRMs are obtained by flattening (which proves in other way why there is  $2^n$  of such forms). If for every variable one uses either rule 5, rule 6, or rule 7, the KRM Trees are created, from which KRMs are obtained by flattening (which proves in other way why there is  $3^n$  of such forms). If rules 5, 6 and 7 are used, but in each subtree there is a choice of a

rule, the PKRM Trees are generated from which PKRM forms are obtained by flattening. Now, if additionally we allow the expansion variables to have various orders (but the same in the entire tree), one obtains the QKRM Trees, and PKRM flattened forms, respectively. One can now see that a further natural generalization is to allow various orders of variables in subtrees of QKRM trees to create an even wider family of trees. There are two ways of generalizing those forms for the logic with multiple-valued inputs. One was shown in [19]. The other one will be presented here.

### 3 Generalizations of Reed-Muller Forms for the Logic with Multiple-Valued Inputs

**Definition 3.1.** A multiple-valued input, two-valued output, completely specified switching function  $f$  (*multiple-valued function*, for short) is a mapping:  $f(X_1, X_2, \dots, X_n): P_1 \times P_2 \times \dots \times P_n \rightarrow \{0,1\}$ , where  $X_i$  is a *multiple-valued variable*,  $P_i = \{0, 1, \dots, p_i - 1\}$  is a *set of truth values* that this variable may assume. This is a generalization of an ordinary  $n$ -input switching function  $f: \{0,1\}^n \rightarrow \{0,1\}$ .

**Definition 3.2.** For any subset  $S_i \subseteq P_i$ ,  $X_i^{S_i}$  is a *literal* of  $X_i$ . The set of values  $S_i$  will be called the *polarity of literal*  $X_i^{S_i}$ . The literal  $X_i^{S_i}$ , where  $S_i \in P_i$  is defined as follows:  $X_i^{S_i} = 1$  if  $X_i \in S_i$ ; and  $X_i^{S_i} = 0$  otherwise.

**Example 3.1.** For values 0,1 or 2 of a 5-valued variable  $X$ , the literal  $X^{0,1,2}$  equals 1. For values 3 or 4 of a 5-valued variable  $X$ , the value of the literal  $X^{0,1,2}$  equals 0.

**Definition 3.3.** A product of literals,  $X_1^{S_1} X_2^{S_2} \dots X_n^{S_n}$ , is referred to as a *product term* (also called *term* or *product* for short). A sum of products is denoted as a (multi-valued input) *sum-of-products expression (SOPE)*.

**Example 3.2.** 2-bit decoders have pairs of primary inputs of the function as their inputs. Assume pairing of variables  $X_1 = (x_i, x_j)$ . The corresponding 2-bit decoder has two input variables;  $x_i$  and  $x_j$ , and  $2^2 = 4$  outputs:  $\bar{x}_i + \bar{x}_j$ ,  $\bar{x}_i + x_j$ ,  $x_i + \bar{x}_j$ ,  $x_i + x_j$ . Those outputs correspond to the following literals of variable  $X_1$ :  $X_1^{0,1,2}$ ,  $X_1^{0,1,3}$ ,  $X_1^{0,2,3}$ ,  $X_1^{1,2,3}$ , respectively.

Switching functions with multiple-valued inputs, two-valued outputs, find several applications in logic design, pattern recognition, and other areas. In logic design, they are primarily used for the minimization of PLAs that have 2-bit decoders on the inputs. A Programmable Logic Array (PLA) with  $r$ -bit input decoders directly realizes a SOPE of a  $2^r$ -valued input, two-valued output, function [25]. Such decoders can be also used in any other realization of the logic with multiple-valued inputs, like multiple-valued input ESOPs [18,27]. A simplified form of such decoders was used in [29,30,31] in the realization of *Multiple-Valued Input Kronecker Reed-Muller Forms (MIKRM<sub>s</sub>)*. It will be also used in the "fixed-polarity" *Multi-Valued Input Kronecker Reed-Muller Trees (MIKRM<sub>MT</sub><sub>s</sub>)* that will be introduced here. This simplification consists in creating a simplified decoder with  $2^r - 1$  outputs for  $r$  input signals. The set of outputs of the simplified decoder is a subset of functions (all but one) realized by a standard decoder. For instance, for a 4-valued input signal  $X$  one needs any three outputs of a standard decoder from Example 3.2.



In the case of binary-input logic, each variable  $x_i$  from a GRM form can have one of two possible *polarities*, 0 or 1. The notation used for binary functions is:  $x_i^0 = \bar{x}_i$ ,  $x_i^1 = x_i$ . Let us observe that if two polarities were available for even a single variable, then the ESOP expression including literals of both polarities would be not canonical, for instance  $x$  and  $1 \oplus \bar{x}$  would represent two different expressions for the same function  $f(x) = x$ .

The question arises, how to create canonical generalized Reed-Muller forms for multiple-valued input logic. Methods were shown in [19,29,30,31]. Here we will present another method, that allows for more general interpretations. It is next used to create the family of canonical forms and trees. Let us first observe that in a logic with a  $p_i$ -valued input  $X_i$  there exists  $p_i$  different single logic values: for variable  $X_i$  one can create  $p_i$  different literals with arbitrary single-value polarities. It is obvious that if we will take all those literals to the ESOP expression, then there will be more that one way to describe any Boolean variable function of a single variable. If a single literal from this set of literals is removed, then the remaining literals describe any single-input function in an univocal (canonical) way. For instance, for a  $p$ -valued variable  $X$  one has the literals:  $X^{i_0}, X^{i_1}, X^{i_2}, \dots, X^{i_{p-1}}$ . Removing any one of them, say  $X^2$ , one gets the following literals:  $X^0, X^1, X^3, X^4, \dots, X^{p-1}$ . Such literals will be called *allowed literals*. Literal  $X^2$  is univocally created as  $1 \oplus X^0 \oplus X^1 \oplus X^3 \oplus \dots \oplus X^{p-1}$ . It can be proven [29,30,31] that for a GRM expansion one can take any  $p-1$  single-valued literals, and moreover, any  $p-1$  literals that form an orthogonal *polarity matrix*. For instance for  $p=4$  one can have the following set of allowed literals:  $\{X^{0,1,2}, X^{0,1,3}, X^{0,2,3}\}$ , which is described by a polarity matrix:

$$PM(X) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} X^{0,1,2} \\ X^{0,1,3} \\ X^{0,2,3} \end{bmatrix} \quad (8)$$

It is assumed that logic value 1 (universe) is available. This corresponds to a literal with all possible values, which in turn means a row of all ones in the "expanded" polarity matrix.

The orthogonal expanded polarity matrix includes also a row of ones which corresponds to the universe 1. For the above example the expanded polarity matrix is:

$$EPM(X) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} X^{0,1,2} \\ X^{0,1,3} \\ X^{0,2,3} \\ X^{0,1,2,3} \end{bmatrix} = \begin{bmatrix} X^{0,1,2} \\ X^{0,1,3} \\ X^{0,2,3} \\ 1 \end{bmatrix} \quad (9)$$

Let us observe that all possible literals can be created by exoring rows of this matrix. The *Expanded Polarity Matrix* of variable  $X_i$  is also called *polarity* of this variable. Let us observe that there are the following expanded polarity matrices of binary variables:

$$\text{POLARITY-2-1-A: } EPM(X) = \begin{bmatrix} 1 \\ X^0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\text{POLARITY-2-1-B: } EPM(X) = \begin{bmatrix} 1 \\ X^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\text{POLARITY-2-2: } EPM(X) = \begin{bmatrix} X^0 \\ X^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Let us observe that when all variables are in polarity POLARITY-2-1-B the function is in the binary Reed-Muller form. When all variables are in polarity POLARITY-2-1-A the function is in the binary Negative Reed-Muller form. When all variables are in polarity POLARITY-2-2 the function is in the binary canonical AND-EXOR minterm form. When each variable is either in polarity POLARITY-2-1-A or in polarity POLARITY-2-1-B, the function is in a GRM form. Applying expansions of variables according to POLARITY-2-1-A or polarity POLARITY-2-2, we obtain a new (mixed polarity) canonical form. Similarly, applying expansions of variables according to POLARITY-2-1-B or polarity POLARITY-2-2, we obtain another new (mixed polarity) canonical form. Applying expansions of variables according to POLARITY-2-1-A, POLARITY-2-1-B or polarity POLARITY-2-2, we obtain the Kronecker Reed-Muller canonical form [4,8]. The concept of the polarity matrix will allow now to generalize the concept of canonical trees and forms to the logic with multiple-valued inputs.

Any form in which all variables are in the same polarity is called a Multiple-Valued Input, Binary Output Restricted GRMs (MIRGRM) form. Such form is canonical since the expansion is unique for each of its variables. It can be shown that for a logic with 3-valued inputs there are 29 various polarities, and 29 MIRGRMs. The number of MIRGRM forms for a logic with  $p$ -valued inputs can be calculated from the known mathematical results on the number of orthogonal zero-one matrices. Assuming that universe 1 is available (which is reasonable for practical reasons), expansions that use row of ones in expanded polarity matrix are more interesting. Under such assumption, some examples of sets of allowed literals for a 4-valued input variable  $X$  are:  $\{X^{0,1,2}, X^{0,1,3}, X^{1,2,3}\}$ ,  $\{X^{1,3}, X^{2,3}, X^3\}$ ,  $\{X^{0,2}, X^{0,1}, X^{0,1,2}\}$ ,  $\{X^{1,3}, X^{2,3}, X^{1,2,3}\}$ . It can be easily checked that for all those sets a complete set of all literal values can be obtained from other allowed literals by exoring rows of the expanded polarity matrix. There are examples of using switching functions with such literals for practical circuits such as adders [29,30,31].

Reed-Muller forms are extremely easily testable [6,21,23]. We have proved in a forthcoming paper that also all the generalized binary (and even multiple-valued) Reed-Muller forms discussed here have very good testability properties. Among MIRGRMs for 4-valued logic especially preferable is the form which corresponds to the set of allowed literals:  $\{X^{1,3}, X^{2,3}, X^{1,2,3}\}$ , since the decoder is very simple - a single OR gate:  $x_1 = X^{1,3}$ ,  $x_2 = X^{2,3}$ ,  $x_1 + x_2 = X^{1,2,3}$ . The test generation for this form is easy (it uses an adaptation of methods from the literature). It minimizes the total layout area comparing to other decoders, because of small area of the OR gate.

*Definition 3.4.* The set of allowed literals for a  $p$ -valued variable  $X$  is a set with  $p - 1$  elements whose corresponding polarity matrix is orthogonal.

*Definition 3.5.* Allowed literal is a literal with the set of values corresponding to a row of an orthogonal polarity matrix.

*Definition 3.6.* Polarity vector  $PV = [PM_1, PM_2, \dots, PM_n]$  is a vector of polarity matrices of input variables.

*Definition 3.7.* By a Multiple-Valued Input Kronecker Reed-Muller (MIKRM) Expression for a polarity vector one understands an exclusive sum of products in which all (multiple-valued) literals are allowed literals for this polarity vector.

It can be proven [31] that MIKRM expression is canonical (which means that if for each variable a single polarity is selected, then there exists only one MIKRM expression for this set of variables and their corresponding polarities). It results directly from the fact that for each of its input variables there exists a unique expansion. Therefore we will refer from now on to a MIKRM form, remembering that there are many such forms for a function. Since for ternary variables there are 29 polarities, there are  $29^n$  MIKRMs for a function of  $n$  ternary variables. If the universe is not a row of an EPM(X) then all terms of MIKRM need to include literals of variable  $X$ . This is of course of only theoretical interest, since in the existing technologies the universe (logic constant 1) is available at no cost.

As we can see, the MIKRM form is a generalization of the concepts of GRM and KRM forms. It can be observed that there are no separate generalizations of GRMs and KRMs for the logic with multiple-valued inputs.

It results from the above definitions that the MIKRM class is properly included in the ESOP class. The introduced above concepts and definitions will be now illustrated with an example.

*Example 3.3.* Assuming 4-valued input variables  $X$  and  $Y$ , the expression:

$$f(X, Y) = 1 \oplus X^{0,1,2} Y^{0,1,2} \oplus X^{0,1,3} Y^{2,3} \oplus X^{1,2,3} Y^{2,3} \oplus X^{0,2,3} Y^{1,2,3}$$

is an ESOP but it is not a MIKRM form because there exists the variable  $X$  that has four different polarities, while only three polarities are allowed for it. The equivalent MIKRM can be obtained by the replacement of the fourth literal of variable  $X$  by an EXOR combination of its another literals:

$$f(X, Y) = 1 \oplus (1 \oplus X^{0,1,3} \oplus X^{1,2,3} \oplus X^{0,2,3}) Y^{0,1,2} \oplus X^{0,1,3} Y^{2,3} \oplus X^{1,2,3} Y^{2,3} \oplus X^{0,2,3} Y^{1,2,3}.$$

The rule  $X^{0,1,2} = 1 \oplus X^{0,1,3} \oplus X^{1,2,3} \oplus X^{0,2,3}$  can be written as: 1111 $\oplus$ 1101 $\oplus$ 0111 $\oplus$ 1011 = 1110. By using the "flattening" Boolean rule the expression  $f(X, Y)$  can be now converted to the exor of products form:

$$f(X, Y) = 1 \oplus Y^{0,1,2} \oplus X^{0,1,3} Y^{0,1,2} \oplus X^{1,2,3} Y^{0,1,2} \oplus X^{0,2,3} Y^{0,1,2} \oplus X^{0,1,3} Y^{2,3} \oplus X^{1,2,3} Y^{2,3} \oplus X^{0,2,3} Y^{1,2,3}.$$

As we can easily verify, this last form is a MIKRM, since all literals are now allowed, and

$$PM(X) = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \quad PM(Y) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

## 4 The Orthogonal Expansion for Multiple-Valued Input Switching Functions

Let us assume that a Boolean function in a form of ESOP is represented as a list of terms called ARESOP. In particular, it can be a set of minterms, or a set of disjoint cubes

representing both a SOP and an ESOP. Our algorithms, however, can assume *arbitrary* ESOP, for any kind of orthogonal expansion.

Let us assume that given is a *vector of expanded polarity matrices*:

$$PV = [PM_1, PM_2, \dots, PM_n].$$

To perform an expansion of an ESOP ARESOP with respect to an expanded polarity matrix  $PM(X_i)$  of variable  $X_i$ , one has to convert every literal of variable  $X_i$  in all cubes from the ARESOP to an EXOR combination of literals that are allowed for this polarity (a universe (a vector of ones) is treated as an allowed literal as well). If a cube CUB of ARESOP has no literal  $X_i^s$  and the universe is absent from the expanded polarity matrix, cube CUB should be first represented as  $1 \cdot CUB$ , and next the universe 1 from it should be converted to the EXOR combination of literals allowed for variable  $X_i$ . (This is a generalization of a binary rule  $a = \bar{a}b \oplus ab$ ). It results from the orthogonal properties of the expanded polarity matrix that such conversion for variable  $X_i$  exists and is unique. Next a one level of flattening is executed and the expression is rearranged to the form with all allowed literals factorized. Below we will illustrate the expansion on an example of a function with ternary variables.

*Example 4.1.*

1. Given is a list ARESOP of disjoint cubes corresponding to expression:

$$X^{0,1}Y^0 \oplus X^1Y^{1,2} \oplus X^2Y^2.$$

2. One has to find expansion with respect to variable  $X$ , with allowed literals 1,  $X^{0,1}$ ,  $X^{0,2}$ . The result of conversion (substitution) is:  $X^{0,1}Y^0 \oplus (1 \oplus X^{0,2})Y^{1,2} \oplus (1 \oplus X^{0,1})Y^2$ .
3. After flattening:  $X^{0,1}Y^0 \oplus Y^{1,2} \oplus X^{0,2}Y^{1,2} \oplus Y^2 \oplus X^{0,1}Y^2$ .
4. After factorizing the allowed literals:  $X^{0,1}(Y^0 \oplus Y^2) \oplus X^{0,2}(Y^{1,2}) \oplus 1(Y^{1,2} \oplus Y^2)$ .

In the next stage similar expansion is done for variable  $Y$ . The expansion uses the respective expanded polarity matrix  $EPM(Y)$ .

Two computer-oriented efficient algorithms to perform this kind of expansion for flat forms are given in [29,30,31] and illustrated with examples there. They do not use flattening and factorizing, however, they cannot be also applied to create tree expansions.

Below we will introduce the basic concept of EXOR type Shannon expansion for the mv logic. As it is well-known, the Shannon expansion theorem has been generalized by Rudell [24] for the mv logic. His expansion is of AND-OR type, to be used for mv SOP synthesis. On the other hand, three generalizations of Shannon theorem for Boolean rings are known [4,9] (rules 5,6 and 7 in section 2). Here we will formulate an expansion that generalizes both Rudell's and Davio's expansions: it is in terms of AND-EXOR expansion, and it is for mv logic.

It can be derived that the orthogonal expansion of function  $f$  with respect to multiple-valued input variable  $X_i$  of expanded polarity matrix  $EPM(X_i)$  can be expressed by the following formula:

$$f = \bigoplus_{X_i^{s_j} \in EPM(X_i)} f_{X_i^{s_j}} X_i^{s_j} \quad (10)$$

where the values of  $f_{X_i^{s_j}}$  are calculated as follows:  $[f_{X_i^{s_j}}]^T = [f_{X_{i,j}}]^T [NP]^{-1}$ ;  $[f_{X_i^{s_j}}]$  is a vector of single-literal orthogonal expansions of literal  $X_i^{s_j}$ ,  $j = 0, \dots, p-1$ ;  $[f_{X_{i,j}}]$  is a vector of single-literal standard expansions of single-value literal  $X_i$ , ( $X_i = j$ ),  $j = 0, \dots, p-1$ ;  $[A]^T$  means matrix  $[A]$  transpose;  $[A]^{-1}$  means matrix  $[A]$  inverse;  $[NP]$  is a normalized polarity matrix, which relates polarities of multiple-valued input literals to single-value literals.

Instead of proving this expansion for a general case we will sketch the proof using another example.

*Example 4.2.* The expanded polarity matrix for ternary variable  $X_i$  is:

$$EPM(X_i) = \begin{bmatrix} X_i^{0,2} \\ X_i^{0,1} \\ X_i^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (11)$$

According to formula 10 the orthogonal expansion for  $EPM(X_i)$  is:

$$f = f_{X_i^{0,2}} X_i^{0,2} \oplus f_{X_i^{0,1}} X_i^{0,1} \oplus f_{X_i^2} X_i^2. \quad (12)$$

We will derive the values of  $f_{X_i^{0,2}}$ ,  $f_{X_i^{0,1}}$ , and  $f_{X_i^2}$ . It holds for non-overlapping literals [24]:  $f = f_{X_i^0} X_i^0 + f_{X_i^1} X_i^1 + f_{X_i^2} X_i^2$  which, with respect to the disjointness of  $X_i^r$ ,  $X_i^s$ ,  $s \neq r$ , gives:  $f = f_{X_i^0} X_i^0 \oplus f_{X_i^1} X_i^1 \oplus f_{X_i^2} X_i^2$ . Then:

$$f = f_{X_i^0} X_i^0 \oplus f_{X_i^1} X_i^1 \oplus f_{X_i^2} X_i^2 = f_{X_i^{0,2}} X_i^{0,2} \oplus f_{X_i^{0,1}} X_i^{0,1} \oplus f_{X_i^2} X_i^2 \quad (13)$$

$$\begin{aligned} f &= (f_{X_i^{0,2}} X_i^0 \oplus f_{X_i^{0,2}} X_i^2) \oplus (f_{X_i^{0,1}} X_i^0 \oplus f_{X_i^{0,1}} X_i^1) \oplus f_{X_i^2} X_i^2 = \\ &= (f_{X_i^{0,2}} \oplus f_{X_i^{0,1}}) X_i^0 \oplus f_{X_i^{0,1}} X_i^1 \oplus (f_{X_i^{0,2}} \oplus f_{X_i^2}) X_i^2 \end{aligned} \quad (14)$$

In matrix form, the equation 13 becomes:

$$[f_{X_i^0} \ f_{X_i^1} \ f_{X_i^2}] \begin{bmatrix} X_i^0 \\ X_i^1 \\ X_i^2 \end{bmatrix} = [f_{X_i^{0,2}} \ f_{X_i^{0,1}} \ f_{X_i^2}] \begin{bmatrix} X_i^{0,2} \\ X_i^{0,1} \\ X_i^2 \end{bmatrix} \quad (15)$$

The relation between the disjoint and non-disjoint literals is given by the equation:

$$\begin{bmatrix} X_i^{0,2} \\ X_i^{0,1} \\ X_i^2 \end{bmatrix} = [NP] \begin{bmatrix} X_i^0 \\ X_i^1 \\ X_i^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i^0 \\ X_i^1 \\ X_i^2 \end{bmatrix} \quad (16)$$

Substituting 16 to 15 one obtains:

$$[f_{X_i^0} \ f_{X_i^1} \ f_{X_i^2}] \begin{bmatrix} X_i^0 \\ X_i^1 \\ X_i^2 \end{bmatrix} = [f_{X_i^{0,2}} \ f_{X_i^{0,1}} \ f_{X_i^2}] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i^0 \\ X_i^1 \\ X_i^2 \end{bmatrix} \quad (17)$$

From there:

$$\begin{aligned}
 [f_{X_i,0,2} \ f_{X_i,0,1} \ f_{X_i,2}] &= [f_{X_i,0} \ f_{X_i,1} \ f_{X_i,2}] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \\
 &= [f_{X_i,0} \ f_{X_i,1} \ f_{X_i,2}] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = [f_{X_i,0} \oplus f_{X_i,1} \quad f_{X_i,1} \quad f_{X_i,0} \oplus f_{X_i,1} \oplus f_{X_i,2}] \quad (18)
 \end{aligned}$$

Formula 18 gives the values  $f_{X_i,0,2}$ ,  $f_{X_i,0,1}$ , and  $f_{X_i,2}$  to be substituted to formula 12 in order to calculate the orthogonal expansion (*End of Example*).

It can be easily checked by substituting respective expanded polarity matrices to formula 10 that the expansions 5 - 7 and the Rudell's expansion are particular cases of this new expansion. This method can be also easily generalized for incompletely specified functions.

1. The orthogonal expansion applied in some restricted way to a multiple-valued input ESOP creates a family of canonical tree expansions analogous to those for binary logic.
2. Applying the expansion uniformly in a tree for a fixed order of expansion variables of the same polarity one obtains the MIRGRM Trees that are the mv counterparts of binary Single Polarity Reed-Muller Trees.
3. Applying the expansion uniformly in a tree for a fixed order of expansion variables of various polarities one obtains the *Multiple-Valued Kronecker Reed-Muller Trees (MIKRM Trees)* that are the mv counterparts of binary GRM Trees and Kronecker Reed-Muller Trees.
4. Applying the expansion in a tree for a fixed order of expansion variables, but having various variable polarities in different sub-expressions (sub-trees) one obtains the *Multiple-Valued Pseudo-Kronecker Reed-Muller Trees (MIPKRM Trees)* that are the mv counterparts of binary Pseudo-Kronecker Reed-Muller Trees.
5. Applying the expansion in a tree for all possible but fixed orders of expansion variables, and having various variable polarities in different sub-expressions (sub-trees) one obtains the *Multiple-Valued Quasi-Kronecker Reed-Muller Trees (MIQKRM Trees)* that are the mv counterparts of binary Quasi-Kronecker Reed-Muller Trees.
6. Applying the expansion in a tree for all possible orders of expansion variables, having various orders in various sub-trees, and having various variable polarities in different sub-expressions (sub-trees) one obtains a new family of canonical trees.
7. The method can be applied with little modification to multi-output functions: it is applied to each function separately. The logically equivalent sub-trees are be combined, which leads to DAG circuits. This transformation preserves the canonicity of the tree circuits.

8. The trees from all above new families of canonical trees can be flattened to respective canonical mv forms. This leads to MIRGRM forms, MIKRM forms, MIPKRM forms, MIQKRM forms, and new mv canonical forms, respectively.

A more detailed characteristics of the above expansions, new mv expansions and computer algorithms to create them will be included in our forthcoming paper.

## 5 Conclusion

In this paper several well-known canonical forms have been generalized for the logic with multiple-valued inputs. An Orthogonal Expansion Theorem has been also formulated, which plays that fundamental a role in those expansions as one played by the Shannon Theorem in inclusive logic and the three Boolean ring expansions for the binary forms. Since the Shannon theorem has several important application in tautology, complementation, implicants generation and many other areas, and the ring expansions are fundamental to EXOR circuits theories, we expect this theorem to play also a fundamental role in the multiple-valued logic.

The reader must bear in mind that the expansions proposed here relate to trees and not "flat" forms. For instance, the GRM forms are independent on the order of variables, but the respective GRM trees do depend on this order. Therefore, investigating expansions with changing the order of variables has practical sense only for some types of expansions. Since several expansions obtained by changing the order of variables produce the same "flat" form, counting of several forms can be difficult, as already observed for Quasi-Kronecker forms by Green [4]. It is even more so for our forms, where different orders of variables in subtrees are possible.

## References

- [1] Ph.V. Besslich, "Efficient Computer Method for EXOR Logic Design", *Proc. IEE*, Vol. 130, Part E, CDT, No. 6., pp. 203-206, 1983.
- [2] D. Brand, T. Sasao, "On the Minimization of And-Exor Expressions", *23 FTC*, pp. 1 - 9, 1990.
- [3] R.K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
- [4] P. Davio, J.P. Deschamps, A. Thayse, *Discrete and Switching Functions*. George and McGraw-Hill, New York, 1978.
- [5] E. Detjens, "FPGA Devices Require FPGA-specific Synthesis Tools", *Computer Design*, p. 124, Nov 1990.

- [6] H. Fujiwara, *Logic Testing and Design for Testability*, Computer System Series, The MIT Press, 1986.
- [7] GOTHIC CRELLON, "The Beginners Guide to Programmable ASICs", 1990.
- [8] D. Green, *Modern Logic Design*, Electronic Systems Engineering Series, 1986.
- [9] D. Green, "Families of Reed-Muller Canonical Forms", *Int. J. Electronics*, Vol. 70, No. 2, pp. 259-280, Jan. 91.
- [10] M. Helliwell, M.A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms", *Proceedings of 25th ACM/IEEE Design Automaton Conference*, 1988, Las Vegas, pp. 427 - 432.
- [11] K.L. Kodandapani, "A Note on Easily Testable Realizations for Logical Functions", *IEEE Trans. Comp.*, Vol. C-23, pp. 332-333, 1974.
- [12] H. Landmann, "Logic Synthesis at Sun", *IEEE conference paper*, CH 2686 - 4 / 89 / 0000 / 0469, 1989.
- [13] MONOLITIC MEMORIES, INC., "XOR PLDs Simplify Design of Counters and Other Devices", *EDN*, May 28, 1987.
- [14] A. Mukhopadhyay, G. Schmitz, "Minimization of Exclusive-OR and Logical Equivalence Switching Circuits", *IEEE Trans. Comp.*, Vol. C-19, No. 2., pp. 132-140, February 1970.
- [15] D.E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection", *IRE Trans. Electron. Comp.*, Vol EC-3, pp. 6-12, September 1954.
- [16] G. Papakonstantinou, "Minimization of modulo-2 sum of products", *IEEE Trans. on Computers*, Vol. C-28, pp. 163-167, February 1979.
- [17] M.A. Perkowski, M. Chrzanowska-Jeske, "An Exact Algorithm to Minimize Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions", *Proc. International Symposium on Circuits and Systems*, May 1990.
- [18] M.A. Perkowski, M. Helliwell, P. Wu, "Minimization of Multiple-Valued Input Multi-Output Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions", *Proc. IEEE Inter. Symp. Multiple Valued Logic*, Guangzhou, People's Republic of China, May 1989, pp. 256-263.
- [19] M.A. Perkowski, P. Dysko, B.J. Falkowski, "Two Learning Methods for a Tree-Search Combinatorial Optimizer", *Proceedings of IEEE International Phoenix Conference on Computers and Communication*, Scottsdale, Arizona, March 1990.
- [20] M.A. Perkowski, M. Chrzanowska-Jeske, "Tree Search Algorithms to Find Exact ESOP Forms", *PSU Report*, 1991.



- [21] D.K. Pradhan, *Fault-Tolerant Computing. Theory and Techniques. Vol. I.*, Prentice-Hall, 1987.
- [22] I.S. Reed, "A Class of Multiple-Error-Correcting Codes and Their Decoding Scheme", *IRE Trans. Inf.Th.*, Vol. PGIT-4, pp. 38-49, 1954.
- [23] S.M. Reddy, "Easy testable realizations for logic functions", *IEEE Trans. Comput.*, Vol. C-21, pp. 1183 - 1188, 1972.
- [24] R. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis", Master Thesis, *University of California*, Berkeley, June 1986.
- [25] T. Sasao, H. Terada, "Multiple-Valued Logic and the Design of Programmable Logic Arrays with Decoders", *Proc. of 9th International Symposium on Multiple-Valued Logic*, Bath, England, pp. 27 - 37, 1979.
- [26] T. Sasao, P. Besslich, "On the Complexity of MOD-2 Sum PLA", *Institute of Electronics and Communication Engineers of Japan*, FTS86-17, pp. 1-8, Nov. 17, 1986.
- [27] T. Sasao, "EXMIN: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions", *Proc. of 20th Int. Symp. on Multiple-Valued Logic*, pp. 128-135, May 1990.
- [28] J.M. Saul, "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representations", *Proc. ICCD '90*, pp. 372-375, Sept. 1990.
- [29] I. Schaefer, "An Effective Cube Comparison Method for Discrete Spectral Transformations of Logic Functions", *M. Sc., Thesis*, May 1990.
- [30] I. Schaefer, M.A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms", *Proc. of the ISMVL-91*, Victoria, B.C., Canada, May 1991.
- [31] I. Schaefer, M.A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms", *submitted to IEE Journal*, May 1991.
- [32] SIGNETICS, PLD Data Manual, Signetics' Approach to Logic Flexibility for the '80's", 1986.
- [33] XILINX, Inc., *"The Programmable Gate Array Data Book"*, 1989.

-----

-----

-----

-----

-----

-----

-----

-----

-----

-----

# Asynchronous Sequential Circuit Design Using Pass Transistor Iterative Logic Arrays

M. N. Liu, G. K. Maki and S. R. Whitaker  
NASA Engineering Research Center  
for VLSI System Design  
University of Idaho  
Moscow, Idaho 83843

*Abstract* - The Iterative Logic Array (ILA) is introduced as a new architecture for asynchronous sequential circuits. This is the first ILA architecture for sequential circuits reported in the literature. The ILA architecture produces a very regular circuit structure. Moreover, it is immune to both 1-1 and 0-0 crossovers and is free of hazards. This paper also presents a new critical race free STT state assignment which produces a simple form of design equations that greatly simplifies the ILA realizations.

## 1 Introduction

A major goal of modern Very Large Scale Integrated (VLSI) design is to produce a structure that consists of similar, if not identical, modules. With such a structure only one module needs to be designed and then can be replicated to realize the final circuit. Very few design procedures have been advanced for sequential circuits that produce structures that are easily realized for VLSI implementations.

This paper introduces an Iterative Logic Array (ILA) architecture for the realization of asynchronous sequential circuits. With the ILA architecture, an asynchronous sequential circuit can be built in a very regular form with a single type of ILA module as a building block. Furthermore, the ILA asynchronous circuits have some unique features, such as immunity to both 1-1 overlapping and 0-0 crossing, tolerant of function hazards and immunity to input bounce. The fundamental mode operation is still required in the ILA asynchronous sequential circuits.

To further simplify the circuit of ILA module, a new state assignment has been developed to generate a new form of design equations (the simple term equation). In the simple term design equations, each coefficient is simply a state variable or a constant, instead of the random sum-of-product expression found in a traditional design equation. That simplifies the basic ILA module to a simple pass logic multiplexer.

## 2 ILA Architecture

Iterative Logic Arrays (ILA) has been described in the literature for quite some time [1,2]. An ILA circuit consists of an array of identical cells. Generally, as shown in Figure 1, each ILA cell contains two sets of input signals. One set of inputs are applied in parallel, while

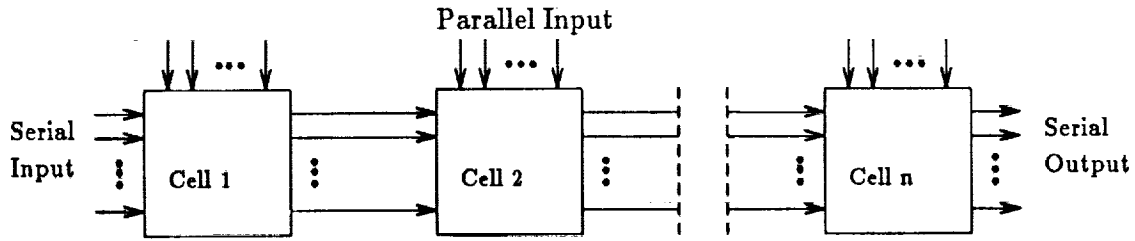


Figure 1: A slice of ILA circuit

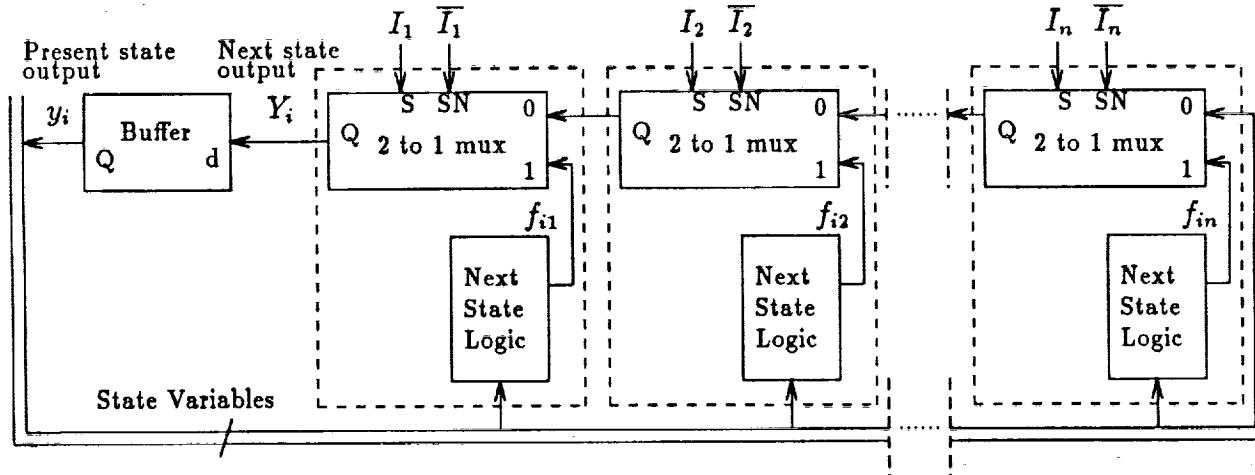


Figure 2: The overall ILA architecture

the other set of inputs are driven by adjacent cells. Signals normally propagate in only one direction between cells, and outputs are derived only from the serial outputs of the last cell. This paper presents an ILA architecture for sequential circuits in which the next state of each state variable is generated by a slice of concatenated ILA cells. A sequential network is then constructed by placing the ILA slices side by side. The function of a flow table is implemented by interconnecting ILA cells and the input states.

The basic cell of an ILA sequential network consists of a 2-to-1 multiplexer (MUX) and a next state forming logic. A MUX cell has a select line  $S$ , its complement  $\bar{S}$  and two data inputs  $I_0$  and  $I_1$ , such that  $Q = S * I_1 + \bar{S} * I_0$ .

The simplest way to implement the MUX function is to use a pass transistor circuit. Basically, the pass transistor MUX, excluding level restoration logic, is a module of two pass transistors, which functions as two simple switches. Design considerations, such as level restoration, are assumed to be handled by the output buffers. The circuit design considerations have been discussed in [3,4,5].

The overall architecture for the ILA sequential circuit is shown in Figure 2 which implements the next state equation

$$Y_i = f_{i1}I_1 + f_{i2}I_2 + \cdots + f_{in}I_n. \quad (1)$$

**Theorem 1** *The architecture depicted in Figure 2 is a proper model for an asynchronous sequential circuit.*

**Proof:** It is assumed that an STT assignment is used and the logic equations are defined in Equation 1. Let the present input be  $I_p$ , which means that only MUX cell  $p$  passes the next logic state  $f_{ip}$  to the buffer. Therefore,  $Y_i = f_{ip}I_p$  for each  $y_i$ . When input  $I_q$  is present ( $I_p = 0, I_q = 1$ ), the only MUX that passes next logic state is cell  $q$ . Therefore  $Y_i = f_{iq}I_q$  for each  $y_i$ . Clearly, the architecture realizes Equation 1.  $\square$

The present state is depicted by the state variables and is fed back to each ILA cell. The logic for each state variable  $Y_i$  consists of  $n$  ILA cells as defined in Figure 2. The set of  $n$  cells is described as a slice that realizes a next state variable. Referring to Figure 2, all ILA cells that are driven by the same input state  $I_i$  belong to the same level. With  $n$  input states and  $m$  state variables, there are  $n$  levels of ILA cells and  $m$  slices.

### 3 Design Procedure for Asynchronous ILA Sequential Network

As a general architecture, the ILA architecture can be used to realize the asynchronous design equations of any STT assignment. This section compares the ILA circuits for traditional STT assignments and proposes a new state assignment which minimizes the next state forming logic in the ILA cell.

#### 3.1 Simple Term Design Equations

The set of next state equations provides a mathematical model of a sequential circuit. For example, the design equations using Liu's assignment for Table 1 are:

$$\begin{aligned} Y_1 &= y_1I_1 + (y_1\overline{y_2} + y_2y_4)I_2 + \overline{y_3}I_3 \\ Y_2 &= y_1I_1 + y_2I_2 + y_3I_3 \\ Y_3 &= 0 + \overline{y_2}I_2 + y_3I_3 \\ Y_4 &= 0 + y_2y_4I_2 + y_3I_3 \end{aligned}$$

If an ILA network is utilized to implement each next state equation, then the next state forming logic  $f_{ip}$  in Equation 1 would be a random logic function in sum of products form. For example,  $f_{12}$  would be  $y_1\overline{y_2} + y_2y_4$ . A circuit to realize the next state forming logic is simply combinational logic and can be formed from NAND gates, NOR gates or from pass transistors.

Some research has been conducted to simplify the next state logic. One such effort was made by Chung-jen Tan [6]. The Tan assignment will produce equations in a form of so called *simple product term equation* in which each coefficient  $f_{ip}$  is simplified to an OR expression, instead of a random sum of products expression.

With traditional state assignments, each  $f_{ip}$  term in resulting design equations is a function of the partitioning variables of input  $I_p$  [7] and the number of partitioning variables

	$I_1$	$I_2$	$I_3$	$y_1$	$y_2$	$y_3$	$y_4$
A	A	B	C	0	0	0	0
B	A	B	G	0	0	1	0
C	D	E	C	1	0	0	0
D	D	F	C	1	1	0	0
E	D	E	G	1	0	1	0
F	A	F	C	0	1	0	0
G	A	H	G	0	1	1	1
H	D	H	C	1	1	0	1

Table 1: A flow table and traditional Liu's assignment.

is equal to the number of k-sets in  $I_p$ , the maximum number of inputs to  $f_{ip}$  logic is equal to the number of k-sets. Therefore, a completely regular ILA circuit for a traditional assignment would contain a k-input next state forming logic for each  $f_{ip}$  where k="number of k-sets in input  $I_p$ ".

A goal of the new design procedure is to minimize the amount of hardware required by the  $f_{ip}$  logic. Physically, the minimal form of  $f_{ip}$  logic is a wire. The design equation in which all  $f_{ip}$  terms are in minimum logic is called a simple term equation.

**Definition 1** *A simple term next state equation is a design equation in which each  $f_{ip}$  coefficient is a single state variable (or its complement) or a constant.*

In a simple term equation, each coefficient depends on at most one state variable. This feature has significant impact on the hardware implementation. With a simple term equation, only a wire is required to generate each  $f_{ip}$  coefficient. With a procedure discussed in later sections, the simple term equations for Table 1 can be derived as follows,

$$\begin{aligned}
 Y_1 &= y_1 I_1 + y_3 I_2 + \overline{y_2} I_3 \\
 Y_2 &= I_1 + y_4 I_2 + y_2 I_3 \\
 Y_3 &= I_1 + y_3 I_2 + 0 \\
 Y_4 &= \overline{y_1} I_1 + y_4 I_2 + \overline{y_2} I_3
 \end{aligned}$$

As all coefficients are simple terms, no extra logic is required to generate each  $f_{ip}$ .

## 3.2 New State Assignment

In this research, partition algebra is used to derive simple term equations and synthesize asynchronous circuits. The new state assignment, called  $\eta$  assignment, is proposed to produce the simple term design equations. A relationship between  $\eta_i^p$  partitions and  $f_{ip}$  coefficients has been presented in the literature [8]. Theorem 2 in [8] shows part of the discovery and has been used to analyze state assignments in predicting hardware.

**Theorem 2** (1) If  $\eta_i^p = \tau_j$ , then  $f_{ip} = y_j$  or  $\overline{y_j}$ . (2) If all the states of  $\eta_i^p$  are in one block, then  $f_{ip} = 0$  or 1.

If all  $f_{ip}$  of the design equations meet the conditions of Theorem 2, then the coefficients will be simple terms. A sufficient condition of  $\eta$ -partitions for simple term  $f_{ip}$  is listed in Corollary 1.

**Corollary 1** *If all  $\eta_i^p$  partitions satisfy one of the following conditions:*

1.  $\eta_i^p = \tau_j$
2.  $\eta_i^p = \{S; \phi\}$
3.  $\eta_i^p = \{\phi; S\}$

*then the design will yield simple terms, where  $S$  is a set of all flow table states.*

**Proof:** The proof follows directly from satisfying Theorem 2.

□

Most STT assignments do not meet the conditions of Corollary 1. However, if an assignment can be generated such that for each  $\eta$ -partition  $\eta_j^p$  under input  $I_p$  where  $\eta_j^p$  does not satisfy the conditions from Corollary 1, a new  $\tau$ -partition  $\tau_k$  will be created where  $\tau_k = \eta_j^p$  to allow  $f_{jp} = y_k$  and produce a simple term equation for  $Y_j$ .

For example, the flow table shown in Table 1 has eight k-sets:

$$\{ABFG\}, \{AB\}, \{ACDFH\}, \{CDEH\}, \{CE\}, \{DF\}, \{BEG\}, \{GH\}.$$

A set of  $\tau$ -partitions can be formed to partition each k-set from the rest of states and the results are  $\tau_1$  through  $\tau_8$  in Table 2 (a). Then from the corresponding  $\eta$ -partitions, it can be found that some  $\eta$ -partitions, such as  $\eta_1^2, \eta_3^2, \eta_4^2$  and  $\eta_7^2$ , are not equal to any  $\tau$ -partitions or a constant. A new  $\tau$ -partition needs to be formed for each of  $\eta$ -partitions which do not meet the conditions of Corollary 1. For each newly created  $\tau$ -partition, corresponding  $\eta$ -partitions need to be formed. The new  $\tau$ -partitions may generate new  $\eta$ -partitions which do not meet any conditions of Corollary 1. Additional  $\tau$ -partitions would then be required. The partitioning procedure will continue until the conditions of Corollary 1 are met. In the case of assignment for Table 1,  $\tau_9, \tau_{10}, \tau_{11}, \tau_{12}$  are formed which in turn generate 12 corresponding  $\eta$  partitions. The results are shown in Table 2 where all  $\eta$ -partitions are equal to a  $\tau$ -partition or a constant.

Simple term equations can be generated once all  $\eta$ -partitions satisfy Corollary 1. In most cases, however, more  $\tau$ -partitions (state variables) than necessary have been introduced and can be removed without jeopardizing the simple term feature.

**Definition 2** *A  $\tau$ -partition  $\tau_i$  is redundant if  $\tau_i$  and  $\eta_i^p$  for all  $I_p$  can be eliminated while the resulting next state equations remain the simple term equations and the state assignment remains a critical race free STT assignment.*

**Theorem 3** *In the set of  $\tau$ -partitions resulting from the  $\eta$  assignment, if a  $\tau_i$ , which partitions a k-set  $K_i$  in  $I_p$  from other states is not equal to any  $\eta_j^p$  or its logical complement, where  $i \neq j$ , then  $\tau_i$  is redundant.*

$\tau_1 = \{ABFG; CDEH\}$	$\tau_2 = \{AB; CDEFGH\}$	$\tau_3 = \{ACDFH; BEG\}$
$\tau_4 = \{CDEH; ABFG\}$	$\tau_5 = \{CE; ABDFGH\}$	$\tau_6 = \{DF; ABCEGH\}$
$\tau_7 = \{BEG; ACDFH\}$	$\tau_8 = \{GH; ABCDEF\}$	$\tau_9 = \{ABCE; DFGH\}$
$\tau_{10} = \{ABDF; CEGH\}$	$\tau_{11} = \{CEGH; ABDF\}$	$\tau_{12} = \{DFGH; ABCE\}$
(a) $\tau$ -partitions		
$\eta_1^1 = \{ABFG; CDEH\}$	$\eta_1^2 = \{ABDF; CEGH\}$	$\eta_1^3 = \{BEG; ACDFH\}$
$\eta_2^1 = \{ABFG; CDEH\}$	$\eta_2^2 = \{AB; CDEFGH\}$	$\eta_2^3 = \{-; ABCDEFGH\}$
$\eta_3^1 = \{ABFGCDEH; -\}$	$\eta_3^2 = \{DFGH; ABCE\}$	$\eta_3^3 = \{ACDFH; BEG\}$
$\eta_4^1 = \{CDEH; ABFG\}$	$\eta_4^2 = \{CEGH; ABDF\}$	$\eta_4^3 = \{ACDFH; BEG\}$
$\eta_5^1 = \{-; ABFGCDEH\}$	$\eta_5^2 = \{CE; ABDFGH\}$	$\eta_5^3 = \{ACDFH; BEG\}$
$\eta_6^1 = \{CDEH; ABFG\}$	$\eta_6^2 = \{DF; ABCEGH\}$	$\eta_6^3 = \{-; ABCDEFGH\}$
$\eta_7^1 = \{-; ABFGCDEH\}$	$\eta_7^2 = \{ABCE; DFGH\}$	$\eta_7^3 = \{BEG; ACDFH\}$
$\eta_8^1 = \{-; ABFGCDEH\}$	$\eta_8^2 = \{GH; ABCDEF\}$	$\eta_8^3 = \{BEG; ACDFH\}$
$\eta_9^1 = \{ABFG; CDEH\}$	$\eta_9^2 = \{ABCE; DFGH\}$	$\eta_9^3 = \{ACDFH; BEG\}$
$\eta_{10}^1 = \{ABCDEFGH; -\}$	$\eta_{10}^2 = \{ABDF; CEGH\}$	$\eta_{10}^3 = \{-; ABCDEFGH\}$
$\eta_{11}^1 = \{-; ABCDEFGH\}$	$\eta_{11}^2 = \{CEGH; ABDF\}$	$\eta_{11}^3 = \{ABCDEFGH; -\}$
$\eta_{12}^1 = \{CDEH; ABFG\}$	$\eta_{12}^2 = \{DFGH; ABCE\}$	$\eta_{12}^3 = \{BEG; ACDFH\}$
(b) $\eta$ -partitions		

Table 2: The  $\tau$  and  $\eta$  partitions

**Proof:** If the k-set  $K_i$  in  $I_p$  is the left block of  $\tau_i$  and the  $\tau_i$  is not equal to any  $\eta_j^p$  or its logical complement, for all  $i \neq j$ , then for every  $I_q, q \neq p$ , there must exist a k-set  $K_k$  under  $I_p$  in the right block of  $\tau_i$  such that the stable state of  $K_i$  and the stable state of  $K_k$  are in the same k-set in  $I_q$ . Hence the k-set  $K_i$  does not need to be partitioned from k-set  $K_k$  for the input transition  $I_p$  to  $I_q$ . Moreover, the partitioning variable  $y_i$  is not needed to generate any simple term equation  $Y_j$ . Therefore,  $\tau_i$  is redundant.

□

For example, in Table 2,  $\tau_2, \tau_5, \tau_6$  and  $\tau_8$  do not appear in  $\eta$ -partition set except for  $\eta_2^2, \eta_5^2, \eta_6^2$ , and  $\eta_8^2$ . Those  $\tau$ -partitions are redundant according to Theorem 3. Another way of looking at this is that  $y_2$  only appears in the equation for  $Y_2$  and no other - same for  $y_5, y_6$  and  $y_8$ . Therefore, they are redundant.

**Theorem 4** If  $\tau_i$  is a logical complement of  $\tau_j$ , then  $y_i$  can be replaced by  $\overline{y_j}$  in all design equations.

**Proof:** If  $\tau_i$  is the logical complement of  $\tau_j$ , then  $\tau_i$  and  $\tau_j$  partition the same k-sets. Only one of them is needed to meet the partitioning condition for a critical race free STT assignment. Moreover, according to the Rule 2 in Theorem 7, all coefficients where  $f_{kp} = y_i$  can be replaced by  $f_{kp} = \overline{y_j}$ .

□



$$\begin{array}{ll} \tau_1 = \{ABFG; CDEH\} & \tau_3 = \{ACDFH; BEG\} \\ \tau_{10} = \{ABDF; CEGH\} & \tau_{12} = \{DFGH; ABCE\} \end{array}$$

(a)  $\tau$ -partitions

$$\begin{array}{lll} \eta_1^1 = \tau_1 & \eta_1^2 = \tau_{10} & \eta_1^3 = \tau_7 \\ \eta_3^1 = 1 & \eta_3^2 = \tau_{12} & \eta_3^3 = \tau_3 \\ \eta_{10}^1 = 1 & \eta_{10}^2 = \tau_{10} & \eta_{10}^3 = 0 \\ \eta_{12}^1 = \tau_4 & \eta_{12}^2 = \tau_{12} & \eta_{12}^3 = \tau_7 \end{array}$$

(b)  $\eta$ -partitionsTable 3: The reduced  $\tau$  and  $\eta$  partitions

Theorem 4 provides the condition for removing another type of redundancy. For example,  $\tau_4$  in Table 2 is the logical complement of  $\tau_1$ . Therefore, equation  $Y_4$  is redundant equation and all coefficients  $y_4$  in the simple term equations can be replaced by  $\overline{y_1}$ . Similarly, equation  $Y_7$  is also a redundant equation and all coefficients  $y_7$  in the simple term equations can be replaced by  $\overline{y_3}$ .

Theorem 4 does not specify which  $\tau$ -partition should be removed if  $\tau_i$  is a logical complement of  $\tau_j$ . The final result may be quite different if  $\tau_i$  or  $\tau_j$  is removed because removing such a  $\tau$ -partition may make more  $\tau$ -partitions become redundant with the condition of Theorem 3. In order to obtain a better result, the redundancy specified by Theorem 3 first needs to be removed. That will allow the designer to make a better choice by checking fewer  $\tau$  and  $\eta$ -partitions. For example,  $\tau_9$  and  $\tau_{11}$  become redundant partitions after removing  $\tau_4, \tau_7$  and corresponding  $\eta$ -partitions  $\eta_4^p, \eta_7^p, \forall p = 1, 2, 3$ .

By eliminating redundant  $\tau$ -partitions and  $\eta$ -partitions, the number of partitions is significantly reduced. In the example above, 8 out of 12  $\tau$  partitions are removed. Table 3 (a) and (b) show the results.

By using Theorem 3, two stable states under the same input state may have an identical state assignment if they have the same next states under all inputs. To have identical assignment for two stable states is equivalent to merge two corresponding rows in the flow table. However, if the outputs associated with these two stable states are not compatible, such a merging is invalid. A unique state has to be assigned to each stable state so that two outputs can be distinguished.

A partitioning chart is introduced to facilitate state assignment reduction and avoid such invalid merging. The chart has intersection for each pair of  $k$ -sets in an input. For each  $\tau$ -partition  $\tau_i$  introduced by the state assignment,  $\tau_i$  will be placed in the intersections where a pair of  $k$ -sets are partitioned by  $\tau_i$ . When a  $\tau_j$  is removed by the state assignment reduction procedure in accordance with Theorem 3,  $\tau_j$  must be removed from all intersections in the partitioning chart.

If a blank intersection in a partitioning chart was left once  $\tau_j$  was removed, then the compatibility of two corresponding rows in the flow table would have to be checked. If

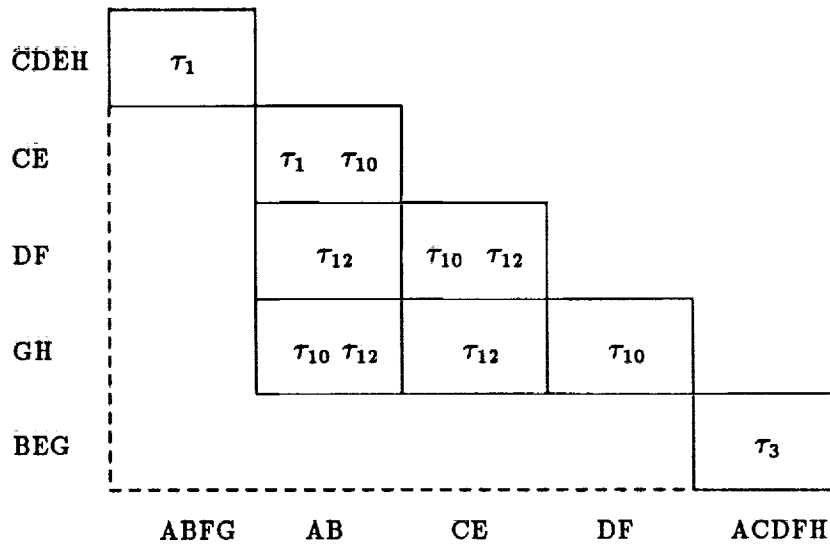


Figure 3: The partitioning chart for the final  $\tau$  partitions

two outputs are not compatible,  $\tau_j$  should not be removed even though  $\tau_j$  is redundant according to Theorem 3. The partitioning chart for the final  $\eta$  assignment is shown in Figure 3. There is at least one  $\tau$ -partition at each intersection.

The following procedure formalizes the  $\eta$  assignment assignment.

**Procedure 1  $\eta$  assignment generation:**

- Step 1.** Select an initial set of  $\tau$ -partition  $\tau_i$  that partition each  $k$ -set  $K_i$  from the rest of states. Create a partitioning chart for  $k$ -sets.
- Step 2.** Generate all  $\eta$ -partitions  $\eta_i^p$  for each input  $I_p$ .
- Step 3.** For any  $\eta_i^p$  that do not meet the conditions of Corollary 1, generate a new  $\tau$ -partition  $\tau_k$  such that  $\tau_k = \eta_i^p$ . Add  $\tau_k$  to the partitioning chart. Return to step 2. The state assignment process is complete when every  $\eta_i^p$  satisfies the conditions of Corollary 1.
- Step 4.** Remove each  $\tau$ -partition  $\tau_i$  that meets the conditions of Theorem 3 if there are no merging problem in the partitioning chart by removing  $\tau_i$ . Also, for each  $\tau_i$  removed, remove  $\eta_i^p$  under all  $I_p$ . Repeat Step 4 until all such  $\tau_i$  have been eliminated.
- Step 5.** For each pair of  $\tau$ -partitions  $\tau_i$  and  $\tau_j$  that are logical complements, remove  $\tau_i$  and  $\eta_i^p$  under all  $I_p$ . Once such a  $\tau_i$  is removed, return to Step 4.

Since Procedure 1 involves iterations of step 2 and step 3, it is useful to know if the number of  $\tau$ -partitions is finite. The following theorem shows closure of Procedure 1.

**Theorem 5** The number of  $\tau$ -partitions needed to generate any arbitrary  $\eta$  assignment is finite.

	$I_1$	$I_2$	$I_3$	$y_1$	$y_3$	$y_{10}$	$y_{12}$
A	<b>A</b>	<b>B</b>	<b>C</b>	1	1	1	0
B	<b>A</b>	<b>B</b>	<b>G</b>	1	0	1	0
C	<b>D</b>	<b>E</b>	<b>C</b>	0	1	0	0
D	<b>D</b>	<b>F</b>	<b>C</b>	0	1	1	1
E	<b>D</b>	<b>E</b>	<b>G</b>	0	0	0	0
F	<b>A</b>	<b>F</b>	<b>C</b>	1	1	1	1
G	<b>A</b>	<b>H</b>	<b>G</b>	1	0	0	1
H	<b>D</b>	<b>H</b>	<b>C</b>	0	1	0	1

Table 4: The result of  $\eta$  assignment

**Proof:** If there are  $k$   $k$ -sets in a column of  $I_p$ , the maximum number of  $\tau$ -partitions that could be generated is

$$\binom{k}{0} + \binom{k}{1} + \binom{k}{2} + \cdots + \binom{k}{k} = 2^k.$$

This is finite.

□

**Corollary 2** *A simple term  $\eta$  assignment exists.*

**Proof:** A flow table must have at least one input state with more than one  $k$ -set, otherwise the circuit is purely combinational. With  $k$   $k$ -set ( $k \geq 2$ ), Theorem 5 shows that the maximum of  $\tau$ -partitions is  $2^k$ . Therefore, an assignment exists because a set of  $\tau$ -partitions can be constructed.

□

**Theorem 6** *The  $\eta$  assignment is a valid STT assignment.*

**Proof:** A Liu assignment will produce  $\tau$ -partitions which partition the  $k$ -sets. The  $\tau$ -partitions in this assignment consists of two sets of partitions. The first consists of the initial set of  $\tau$ -partitions that partitions individual  $k$ -sets. The second set consists of the  $\tau$ -partitions that are created from  $\eta$ -partitions that do not meet conditions of Corollary 1. Hence, both type of  $\tau$ -partitions are elements of a valid Liu assignment and therefore the overall assignment meets the conditions of an STT assignment.

□

The result of  $\eta$  assignment for flow table Table 1 is given in Table 4.

### 3.3 Design Equation Generation

With the  $\eta$  assignment, the next design step is to associate a unique state variable  $y_i$  with each  $\tau$ -partition  $\tau_i$  and determine each  $f_{ip}$  term.

**Theorem 7** *The next state equations can be derived from the  $\eta$ -partitions with the following roles:*

1. If  $\eta_i^p = \tau_j$ , then  $f_{ip} = y_j$ .
2. If the states in the left block of  $\eta_i^p$  are the same as states in the right block of  $\tau_j$  and vice versa, then  $f_{ip} = \overline{y_j}$ .
3. If all the states of  $\eta_i^p$  are in the left block, then  $f_{ip} = 1$ .
4. If all the states of  $\eta_i^p$  are in the right block, then  $f_{ip} = 0$ .

**Proof:** The proof follows directly from Theorem 2 since the  $\eta$  assignment assigns state variable  $y_j = 1$  to states in the left block and  $y_j = 0$  to states in the right block of  $\tau_j$ -partition  $\tau_j$ .

□

The following example gives the next state equations by applying Theorem 7 to the result of  $\eta$  assignment in Table 3.

**Example 1** Applying the revised conditions of Theorem 7 to each  $\eta$ -partition  $\eta_i^p$  in Table 3, the next state equations can be derived as follows:

$$\begin{aligned} Y_1 &= y_1 I_1 + y_{10} I_2 + \overline{y_3} I_3 \\ Y_3 &= I_1 + y_{12} I_2 + y_3 I_3 \\ Y_{10} &= I_1 + y_{10} I_2 + 0 \\ Y_{12} &= \overline{y_1} I_1 + y_{12} I_2 + \overline{y_3} I_3 \end{aligned}$$

The  $\eta$  assignment guarantees to produce the simple term equations in which all coefficients are single variables or constants. The general form for the next state equation is

$$Y_i = \sum_{p=1}^n f_{ip} I_p.$$

It has been shown in [10] that this equation can be represented as a pass logic expression. Since only one  $I_p = 1$  at a time and the case when all  $I_i = 0$  is an undefined situation in an asynchronous flow table, an ILA network will let  $y_i$  be passed to  $Y_i$  if the term  $\overline{I_n}(y_i)$  is added. The equation of  $Y_i$  then becomes

$$Y_i = I_1(f_{i1}) + \overline{I_1}(\cdots I_p(f_{ip}) + \overline{I_p}(\cdots (I_n(f_{in}) + \overline{I_n}(y_i) \cdots)). \quad (2)$$

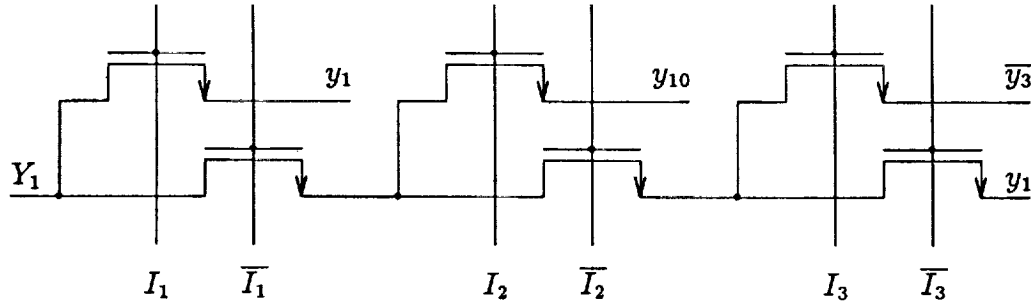
The advantage of this equation is to allow  $Y_i$  to maintain the same state when all  $I_i = 0$ , which can happen during an input transition. For example, the simple term equation for  $Y_1$  in Example 1 is

$$Y_1 = y_1 I_1 + y_{10} I_2 + \overline{y_3} I_3$$

Putting this into the form of Equation 2, the expression can then be converted into

$$Y_1 = I_1(y_1) + \overline{I_1}(I_2(y_{10}) + \overline{I_2}(I_3(\overline{y_3}) + \overline{I_3}(y_1)))$$

Similarly, all other equations can be converted to the pass logic expressions in the same way. The results are as follows:

Figure 4: ILA realization of state variable  $Y_1$ 

$$\begin{aligned}
 Y_1 &= I_1(y_1) + \bar{I}_1(I_2(y_{10}) + \bar{I}_2(I_3(\bar{y}_3) + \bar{I}_3(y_1))) \\
 Y_3 &= I_1(1) + \bar{I}_1(I_2(y_{12}) + \bar{I}_2(I_3(y_3) + \bar{I}_3(y_3))) \\
 Y_{10} &= I_1(1) + \bar{I}_1(I_2(y_{10}) + \bar{I}_2(I_3(0) + \bar{I}_3(y_{10}))) \\
 Y_{12} &= I_1(\bar{y}_1) + \bar{I}_1(I_2(y_{12}) + \bar{I}_2(I_3(\bar{y}_3) + \bar{I}_3(y_{12}))).
 \end{aligned}$$

Obviously, the next state logic  $f_{ip}$  is minimized to a wire if the set of next state equations are all simple term equations. It is straightforward then to map the design equations to an ILA network. Figure 4 shows an ILA realization of state variable  $Y_1$ .

## 4 Input and Hazard Characteristics

In addition to the regularity of the ILA network with the  $\eta$  assignment, an ILA realization has other features such as (1) immunity to input 1-1 overlapping and 0-0 crossing, (2) immunity to input bounce while 1-1 overlapping is not present, and (3) free of transition path hazards and input state transition hazards.

Potential conflicts arise in asynchronous sequential networks when more than one input state is present at a time (1-1 overlapping), or when none of the input states are active (0-0 crossing). The overlapping and crossing situations occur because two input states rarely switch at exactly the same time due to differences in delay through forming logic. Most design procedures avoid such uncertainties by setting a constraint either to forbid 1-1 overlapping or to forbid 0-0 crossing of input states. Another form of hazard on an input signal which may cause the circuit to malfunction is the dynamic hazard when input transitions.

**Theorem 8** *The ILA architecture tolerates 1-1 input overlapping.*

**Proof:** In Equation 2, assume the input state  $I_p$  have higher priority relative to input state  $I_q$ . In the other words,  $I_p$  is the control variable of an ILA cell which is closer to the output than the ILA cell with the control variable  $I_q$ . If both  $I_p$  and  $I_q$  are asserted 1, then  $Y_i$  will assume value specified by  $f_{ip}$  rather than  $f_{iq}$ .

In the case where the input switches from  $I_p$  to  $I_q$ , when  $I_p$  is 1, it passes  $f_{ip}$  to  $Y_i$  and meanwhile cuts off the path of  $f_{iq}$  to  $Y_i$ , no matter if  $I_q$  is set to 1 or 0. With such an

architecture, the 1-1 overlapping of input  $I_p$  and  $I_q$  has the same effect on the output  $Y_i$  as  $I_p = 1, I_q = 0$ .

In the case where the input switches from  $I_q$  to  $I_p$ , when  $I_q = 1$ , all  $Y_i$  are determined by the  $f_{iq}$  values which are propagated through the ILA. Once  $I_p = 1$ , the  $f_{ip}$  values are passed through the ILA independent of whether  $I_q$  is 0 or 1. Hence the circuit assumes the proper next state value.

□

For example, in Figure 4, if  $I_1$  and  $I_2$  are active, the circuit for all next state variables will be under the control of  $I_1$  only, and  $Y_1$  will assume the value of  $y_1$ .  $y_{10}$  can be passed to  $Y_1$  only if  $I_1 = 0$  while  $I_2$  is active. Once  $I_1$  is set to 1 again, the value of  $y_{10}$  at  $Y_1$  will become  $y_1$  immediately (assuming  $\bar{I}_1$  is set to 0 simultaneously).

The 0-0 crossing happens when all input states are 0. Let the input change from  $I_p$  to  $I_q$ . If input  $I_p$  goes to 0 before  $I_q$  goes to 1, there will be a period that all input lines are 0. In traditional design, the outputs of the design equations could assume an undeterminant state when all inputs are 0. The ILA architecture solves the problem by providing a path for each state variable to pass  $y_i$  to  $Y_i$ . When all inputs are 0, it allows  $Y_i$  to maintain its current value  $y_i$ .

Again, Figure 4 can be used to show the feature of 0-0 crossing tolerance. For example, assuming the input transition is from  $I_2$  to  $I_3$ , when  $I_2 = 1$ ,  $y_{10}$  is passed to  $Y_1$ , and  $y_1$  is fed back into the last ILA cell under  $\bar{I}_3$ . Once the network is stable, the level of  $y_{10}$  is passed to  $Y_1$ . When  $I_2$  is set to 0, the path provided by  $\bar{I}_1$ ,  $\bar{I}_2$  and  $\bar{I}_3$  will still maintain the level of  $Y_1$  at the current value of  $y_1$ . The output of the network remains unchanged during the period when all inputs are 0 until  $I_3$  is set to 1. Then a new level of  $y_7$  will be passed to  $Y_1$  and the network assumes a new state.

An input bounce can be considered a dynamic hazard during the transition of input states. With the ability to tolerate input bounce, the ILA network allows extra input transitions to occur before the circuit is stabilized. However, the input bounce can be tolerated only if it is not necessary to also tolerate 1-1 overlapping. If an input  $I_q$  bounce occurs when two input states  $I_p$  and  $I_q$  are overlapping, an ambiguity is created regarding the interpretation of the transition; it could be one transition from  $I_p$  to  $I_q$  or three transitions from  $I_p$  to  $I_q$  then to  $I_p$  back to  $I_q$ . In order to avoid the ambiguity, it is assumed that during the input state transition from  $I_p$  to  $I_q$  the circuit tolerates either the bounce condition or the 1-1 overlapping, but not both.

**Theorem 9** *For the simple term equations derived from an  $\eta$  assignment, all partitioning variables under input  $I_p$  do not change as the circuit transitions from one state to another under  $I_p$ .*

**Proof:** The  $\eta$  assignment produces the simple term equations of the form

$$Y_i = \dots + y_i I_p + \dots$$

where  $y_i$  is the partitioning variable of  $I_p$ .  $y_i$  is the only partitioning variable for all transition paths in the  $\eta$  assignment. According to Theorem 2 in [7],  $y_i$  will not change in

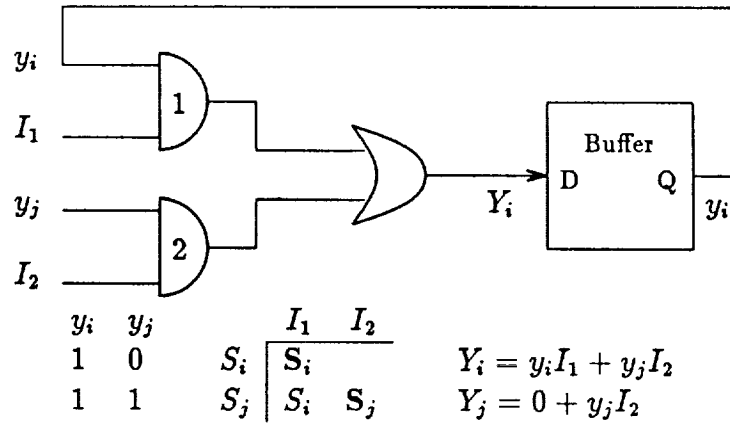


Figure 5: A traditional realization of a partial flow table

any transition under  $I_p$ . Hence, the partitioning variables of the simple term equations do not change during the input transitions.

□

From Theorem 9, the partitioning variables do not change when an input signal  $I_p$  undergoes a bounce. In the case that a dynamic hazard presents when  $I_p$  transitions from 0 to 1, the circuit begins to transition to the new state when  $I_p$  goes to 1. Meanwhile, the partitioning variables of  $I_p$  remain stable. Since the partitioning variables determine all  $f_{ip}$  values [7] and since the partitioning variables are unaffected by the input bounce, the circuit will assume the proper next state value when  $I_p$  is stabilized.

In the case that a dynamic hazard presents when  $I_p$  transitions from 1 to 0, the present state is simply passed to the next state once  $I_p$  goes to 0, as this represents a 0-0 crossover condition. The circuit does not transition. When  $I_p$  returns to 1 on the bounce condition, again from Theorem 9, there will be no transition in partitioning next state variables. Since the circuit is a function only of the partitioning variables [7], output of the circuit remains unchanged.

A critical race free asynchronous sequential network may still malfunction due to unwanted switching transients in the combinational circuit. A transition path hazard is one that is present within the states of a transition path. The simple term equations for the next state variables have the form

$$Y_i = \dots + y_i I_p + \dots$$

where  $y_i$  is the partitioning variable of  $I_p$ . From Theorem 9, partition variable  $y_i$  does not change as the circuit transitions from one state to another in  $I_p$ . In general, since the partitioning variables of the simple term equations do not change during the transition, it is impossible for a hazard of any kind to occur. Therefore, there cannot be any static, dynamic and function hazards that occur during the transition between unstable and stable states.

A circuit free of static and dynamic hazard may have a hazard problem caused by the change of more than one variable in the design equation. One type of such multi-variable change hazard is a function hazard. The problem can be illustrated by the partial flow

table in Figure 5 where design equations for  $Y_i$ ,  $Y_j$  and a schematic for  $Y_i$  are shown. A hazard exists when an input changes from  $I_2$  to  $I_1$ . If the delay of AND gate-1 is longer than the total delays of AND gate-2, the OR gate and the buffer, then the output of gate-1 will remain 0 after the input changes to  $I_1 = 1$  and  $I_2 = 0$ . That will in turn cause  $y_i$  to remain at 0 and lock the output of gate-1 to 0. The result is that state variables  $y_i y_j$  are set to 00 instead of intended value 10. Moreover, the circuit is locked up in the wrong state until a reset line is provided.

The a problem occurs in a traditional design where a slow gate has the same effect as a 0-0 crossover. Input  $I_p = 0$  will set the product term to 0. The ILA circuit solves the problem by maintaining the same state when all inputs go to 0. Also, the 1-1 overlapping problem which may arise in the traditional design due to a slow gate can be tolerated by ILA circuit. The 1-1 overlapping and 0-0 crossover properties of the ILA architecture prevents the circuit from malfunction due to such input state transition hazards.

## References

- [1] C. Roth, *Fundamentals of Logic Design*, 3rd Ed., St. Paul, Minn., West Publishing, 1985.
- [2] D. Givone, *Introduction to Switching Circuit Theory*, McGraw-Hill, Inc., 1970.
- [3] S. Whitaker, "Design of Asynchronous Sequential Circuits Using Pass transistors," Ph.D Dissertation, University of Idaho, Feb. 1988.
- [4] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp. 238-242.
- [5] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78.
- [6] C. Tan, "State Assignments for Asynchronous Sequential Machines", IEEE Transactions on Computers, Vol. C-20, No. 4, April 1971, pp. 382-391.
- [7] S. Gopalakrishnan, G. Kim and G. Maki, "Implications of Tracey's Theorem to Asynchronous Sequential Circuit Design", The 2nd NASA SERC Symposium on VLSI Design, November, 1990, pp. 9.1.1-9.1.11.
- [8] G. Maki, D.Sawin and B. Jeng, "Improved State Assignment Selection Tests", IEEE Transactions on Computer, Dec. 1972, pp.1443-1449.
- [9] J. Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, Aug. 1966, pp. 551-560.
- [10] S. K. Gopalakrishnan, "Design of VLSI Asynchronous Sequential Machines," Ph.D Dissertation, University of Idaho, December, 1989.



# Pulse Mode VLSI Asynchronous Circuits

Q. Chen and G. Maki

NASA Space Engineering Research Center  
for VLSI System Design  
College of Engineering  
University of Idaho  
Moscow, Idaho 83843

*Abstract* - A new basic VLSI circuit element is presented that can be used to realize pulse mode asynchronous sequential circuits. A synthesis procedure is developed along with an unconventional state assignment procedure. Level input asynchronous sequential circuits can be realized by converting a regular flow table into a differential mode flow table, thereby allowing the new synthesis technique to be general. The new circuits tolerate 1-1 crossovers. This circuit also provides a means for state sequence detection and real time fault detection.

## 1 Introduction

Many asynchronous sequential circuits can be modeled as a pulse mode circuit since the inputs are presented in the form of pulses [1]. Level input sequential circuits can be modeled as a pulse mode circuit by detecting input state changes [2]. This work presents a basic circuit that can be used to realize state variables that are effective in the realization of pulse mode circuits.

Sequential circuits are normally defined in terms of flow tables, such as shown in Table 1. The inputs are shown across the top and the states along the side. The states are encoded with internal state variables  $y_i$ . Next state variables  $Y_i$  identify the next state that the circuit will assume.

This paper presents a VLSI circuit element that allows for efficient realizations of pulse mode asynchronous sequential circuits. The network consists of pass transistor next state forming logic with a unique buffer.

The paper describes the following:

- Synthesis procedures for pulse mode asynchronous sequential circuits.
- State assignment procedure for differential mode asynchronous sequential circuits.
- Tolerance of 1-1 input crossover situations. (This circuit is designed to tolerate 0-0 input crossover situations also.)
- State sequence detection.
- Real time fault detection.

## 2 Pulse Mode Circuits

The next state equations can be expressed as follows[3]:

$$Y_i = f_{i1}I_1 + f_{i2}I_2 \cdots + f_{in}I_n \quad (1)$$

where  $Y_i$  is next state variable,  $I_p$  is the input state and  $f_{ip}$  is a sum-of-products expression of state variables. It has been shown that the next state equations can be expressed as a pass logic expression[3]:

$$Y_i = I_1(f_{i1}) + I_2(f_{i2}) \cdots + I_n(f_{in}) \quad (2)$$

where  $I_p(f_{ip})$  means input  $I_p$  passes function  $f_{ip}$ .

The basic circuit to implement pulse mode circuits is shown in Fig. 1. Each state variable is realized with this circuit. If there are  $m$  state variables, then there would be  $m$  such circuits except that there is only one NOR gate.

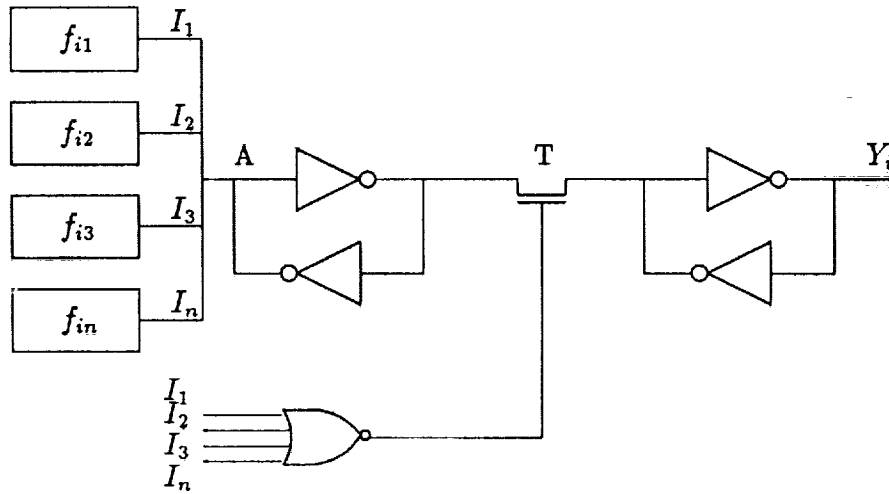


Figure 1: Next State Circuit Module

In pulse mode operation, all  $I_p$  could be 0. When all input states  $I_p$  are 0, the pass networks  $f_{ip}$  are disabled and hence are tristated from the inverter input of the first stage. The feedback inverter in the first stage is provided to sustain the value at point A of the first stage. However, the feedback inverter consists of weak devices that can be overdriven by the  $I_p(f_{ip})$  networks. The same kind of inverter is placed in the second stage of the circuit after transistor T.

For pulse mode operation, assume one and only one input state  $I_p$  is 1 at a time or all  $I_p$  are 0. In other words, only one input pulse is present at a time. When  $I_p = 1$ , pass network  $f_{ip}$  presents the proper next state value to  $\hat{Y}_i$  as specified in Eq. 2 for  $Y_i$  to the input of the inverter at point A. The feedback inverter is composed of weak pullup and pulldown transistors such that they can be overdriven by the value passed by  $I_p(f_{ip})$ . Therefore the correct next state value as defined by Eq. 2 is present at point A in Fig. 1

and  $\hat{Y}_i$  contains the complement of  $Y_i$ . When all  $I_p = 0$ , transistor T is enabled and  $\hat{Y}_i$  is passed to  $Y_i$  and the circuit assumes the proper next state.

To summarize, when one  $I_p = 1$ ,  $\hat{Y}_i$  assumes the complement of the proper next state value of  $Y_i$  as defined by Eq. 2. When all  $I_p = 0$ ,  $\hat{Y}_i$  is passed to the second stage of the inverter and  $Y_i$  assumes the value defined by Eq. 2. The new present state feeds back to the  $f_{ip}$  networks to generate the new next state values to the first stage, dependent on which  $I_p = 1$ . An interesting observation can be made which is common to all asynchronous sequential circuits, but perhaps is more easily seen here. When all  $I_p = 0$ , the present state, as determined by present state variables  $y_i$ , feed back to the  $f_{ip}$  logic. All possible next states are generated and appear at the input of the pass transistors controlled by  $I_p$ . The circuit has "calculated", as determined by Eq. 2, all possible next states that the circuit could enter and is prepared to assume any and every next state as defined by the  $f_{ip}$  terms. The exact next state is specified by the  $I_p$  state that becomes 1.

The state assignment problem for asynchronous sequential circuits is always a significant problem. Pulse mode flow tables are in every way asynchronous in operation. Therefore, the designer must be concerned about state assignment issues. Assume the present state of the circuit is  $S_i$  and state  $S_j$  is the next state when input  $I_j$  becomes 1. When all inputs are 0 prior to  $I_j = 1$ , the state variables  $y_i$  define the circuit to be in state  $S_i$ . When  $I_j = 1$ , since transistor T is disabled, the next state variables  $Y_i$  do not change.  $\hat{Y}_i$  changes to assume values associated with  $S_j$  as defined by Eq. 2 when  $I_j = 1$ . However,  $Y_i$  remains unchanged as long as  $I_j = 1$ .  $Y_i$  does not change to the value of  $S_j$  until  $I_j$  returns to 0, at which time  $\hat{Y}_i$  cannot change. Therefore, each state transition occurs in two stages:

$$\begin{aligned} \hat{Y}_i &= \overline{\sum I_p(f_{ip})} & \text{when } I_p = 1 \\ Y_i &= \hat{Y}_i & \text{when all } I_p = 0 \end{aligned}$$

A critical race can exist in an asynchronous sequential circuit only when the state variables  $y_i$  being fed back can affect  $Y_i$  without a change in input. Since the inputs must change before present state variable  $y_i$  can affect next state variable  $Y_j$ , no critical race can occur. The following theorem has been established.

**Theorem 1** *Asynchronous sequential circuits implemented with the basic circuit shown in Fig. 1 are void of critical races.*

If the circuit cannot experience a critical race, then the Single Transition Time (STT) state assignment procedures need not be followed, specifically the Tracey conditions[5] need not be met. Moreover, since the STT conditions need not be met, any state assignment is satisfactory as long as each state has a unique code.

The design procedure can be stated as follows:

**Procedure 1 Step 1** *Create an appropriate flow table.*

**Step 2** *Provide a state assignment where each state has a unique code.*

$y_1$	$y_2$		XC				Z
			00	01	11	10	
0	0	A	-	A	-	B	0
0	1	B	-	C	-	-	0
1	1	C	-	A	-	D	1
1	0	D	-	C	-	-	1

Table 1: Example Flow Table

**Step 3** Form the state table.

**Step 4** Find the next state equations in the following form:

$$Y_i = \sum I_p(f_{ip})$$

where each input passes an  $f_{ip}$  expression of state variables.

**Example 1** Realize a circuit which has two pulse inputs  $X$  and  $C$  and a level output  $Z$ .  $C$  represents a clock that produces pulses at a regular interval.  $Z$  must be 1 between pulses  $C_i$  and  $C_{i+1}$  only if an  $X$  pulse occurred between clock pulses  $C_{i-1}$  and  $C_i$ .

The reduced flow table with the state assignment is shown in Table 1. The design equations for this flow table are:

$$\begin{aligned} Y_1 &= X(y_1) + C(\overline{y_1}(y_2) + y_1(\overline{y_2})) \\ Y_2 &= X(\overline{y_1}) + C(\overline{y_1}(y_2) + y_1(\overline{y_2})) \end{aligned}$$

## 2.1 Design By Inspection

The synchronous state assignment procedure allows for a great deal of flexibility. The one-hot-code is well known as a state assignment that allows one derive the design equations by inspection. A one-hot-code encodes an  $n$ -row flow table with  $n$ -state variables where state  $S_i$  is encoded with  $y_i = 1$  and all other  $y_j = 0$ ,  $j \neq i$ . A predecessor state of state  $S_i$  is a state the circuit is in prior to an input change that forces the circuit into  $S_i$ .

If  $S_j$  is a predecessor state to  $S_i$  under input  $I_p$ , the partial next state equation is

$$Y_i = I_p(y_j).$$

If  $S_{j1}, S_{j2}, \dots, S_{jk}$  are predecessor states to  $S_i$ , then the partial next state equation is

$$Y_i = I_p(y_{j1} + y_{j2} + \dots + y_{jk}).$$

In general, the  $f_{ip}$  terms become simple sum-of-products where each product is an uncomplemented state variable.

Design Procedure 1 can be employed by simply changing Step 2 to implementing a one-hot-code. The equations can be formed by the well known inspection method. Simpler  $f_{ip}$  terms result. The disadvantage is that more state variables are generally needed. The design equations for Table 1 are

$$\begin{aligned}
Y1 &= C(y_1 + y_3) \\
Y2 &= X(y_1) \\
Y3 &= C(y_2 + y_4) \\
Y4 &= X(y_3)
\end{aligned}$$

Liu[6] proposed a design technique for iterative logic array synchronous sequential circuits that have the unique property where each state has predecessor states only in one input. The next state equations have the form

$$Y_i = I_p(f_{ip})$$

where each  $f_{ip}$  is a sum-of-products with each product term consisting of a single complemented or uncomplemented state variable. This technique reduces the amount of logic further for each next state variable in that only one input  $I_p$  pass gate is needed. The potential disadvantage is that more state variables can be needed.

### 3 Tolerance to 1-1 Input Overlap

In the previous section there were no constraints on the width of each input pulse. (The minimum width must be long enough to pass the signal to the output of the input inverters at the first state). It was assumed that only one  $I_p$  would be 1. This condition can be relaxed. For simplicity, suppose two inputs  $I_p$  and  $I_q$  are both 1. Moreover, suppose the circuit should transition from  $S_i$  to  $S_p$  or  $S_q$  under  $I_p$  or  $I_q$  respectively. When both  $I_p$  and  $I_q$  are 1,

$$\hat{Y}_i = \overline{I_p(f_{ip}) + i_q(f_{iq})}$$

As long as  $I_p$  and  $I_q$  are 1, there can be conflicting signals at the input to the inverter of the first stage of Fig. 1. Since, at least one input = 1, transistor T is not enabled and  $y_i$  does not change and the conflict does not affect the present state. The circuit remains in state  $S_i$  and will remain in  $S_i$  until both  $I_p$  and  $I_q = 0$ . If  $I_p(I_q)$  remains 1 longer than  $I_q(I_p)$ , then  $f_{ip}(f_{iq})$  will be passed to specify  $\hat{Y}_i$  and only when both inputs are 0 will the circuit transition to  $S_p(S_q)$ . Therefore, the circuit action is determined by the input that remains 1 last.

**Theorem 2** *If more than one input state is 1, then the next state of the circuit is determined by the input that remains 1 last.*

**Proof:** If more than one input = 1, then  $\hat{Y}_i$  is determined by the equation

$$\hat{Y}_i = \overline{\sum_{j=t}^k I_j(f_{ij})}$$

where  $I_j$  are those inputs that are 1. Since  $y_i$  changes only when all  $I_j$  are 0, the circuit does not transition until all  $I_j = 0$ . Suppose  $I_p$  is the last input that is 1. Then the equation for  $\hat{Y}_i$  becomes

$$\hat{Y}_i = \overline{I_p(f_{ip})}.$$

When  $I_p$  transitions to 0, then  $Y_i$  assumes the state determined by  $\hat{Y}_i$  which was specified by  $I_p$ .  
QED.

From Theorem 2, it is clear that the order in which inputs transition from  $1 \rightarrow 0$  is important. Transitions from  $0 \rightarrow 1$  are unimportant. Therefore, if more than one input state is 1, it is unimportant which order the inputs transition  $0 \rightarrow 1$ . The next state is specified by the last input that transitions  $1 \rightarrow 0$ . For example, suppose there are four input states for a circuit. If the inputs transition as shown in Fig. 2, the circuit will assume the state specified by  $I_3$  when all the inputs are 0.

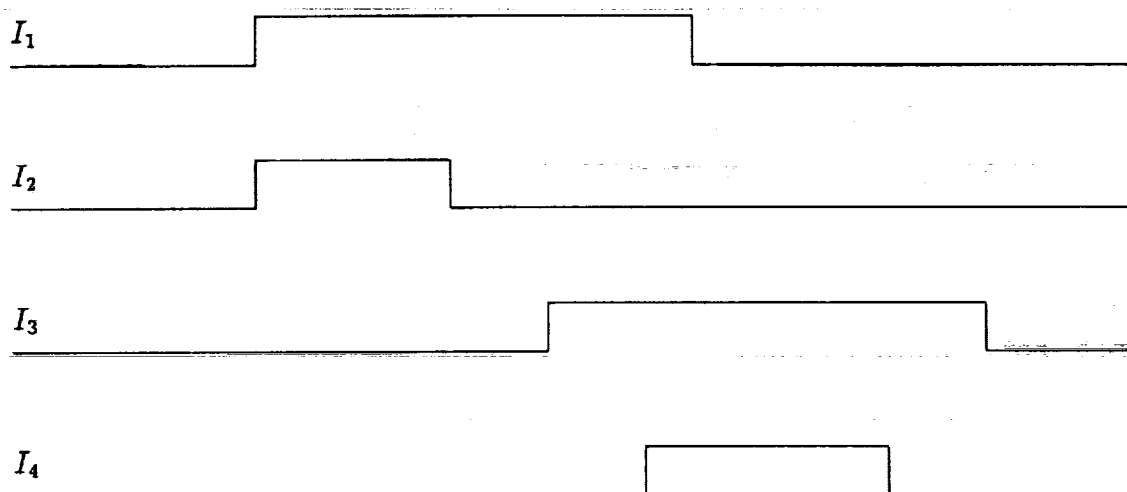


Figure 2: Input Waveform Example

## 4 Level Input Circuits

The previous discussion focused on pulse mode circuits. Several researchers have introduced the notion of transition sensitive asynchronous sequential circuit design [2,7]. Bredeson [2] converted a level input flow table to a transition sensitive (TS) flow table. A TS flow table shows the table entries that result from a change in inputs. The essential feature in a TS design is that inputs are represented as pulses which are created whenever the input state transitions from  $0 \rightarrow 1$ . Consider the level input flow table of Table 2. The TS representation of this flow table is shown in Table 3. Once the flow table is in the TS form, the design procedure in Section 2 applies.

Bredeson introduced another notion in the design of TS circuits. If one begins with a primitive row flow table, then the input state variables can become the state variables. Additional state variables are needed only to produce unique codes for the states and this is accomplished by partitioning stable states in each column of the flow table. In Table 3,

$y_1$	$y_2$	$y_3$		$X_1 \quad X_2$			
				00	01	11	10
0	0	0	A	A	B	-	C
0	1	0	B	A	B	F	G
1	0	0	C	D	H	E	C
0	0	1	D	D	B	E	C
1	1	0	E	D	B	E	G
1	1	1	F	A	H	F	C
1	0	1	G	A	H	E	G
0	1	1	H	D	H	F	C

Table 2: Level Input Flow Table

$y_1$	$y_2$	$y_3$		$X_1 \quad X_2$			
				00	01	11	10
0	0	0	A	C	-	-	-
0	1	0	B	-	F	-	-
1	0	0	C	-	-	-	D
0	0	1	D	C	-	-	-
1	1	0	E	-	-	B	-
1	1	1	F	-	-	H	-
1	0	1	G	-	-	-	A
0	1	1	H	-	F	-	-

Table 3: Transition Sensitive Flow Table

$y_1$  and  $y_2$  are assigned to  $x_1$  and  $x_2$  respectively. State variable  $y_3$  is assigned to partition the stable states in each column. For example  $y_3$  partitions states A and D in the first column. Therefore only one state variable is needed to implement the flow table rather than the expected three.

## 5 State Sequence Detection

It might be desirable to be able to detect the potential transition between a pair of states that might be associated with a critical event. Suppose state  $S_k$  can be entered only from state  $S_i$  under fault free conditions. If state  $S_k$  is entered from state  $S_n$ ,  $i \neq n$ , then an error has occurred. In some cases, such a transition should not be allowed.

The circuit presented here is capable of providing information necessary to detect the occurrence of a transition between a pair of states prior to the actual transition. If one knows that an undesirable transition is about to occur, it is possible to prevent the transition and avoid an unwanted event.

State information is present at two points in the circuit of Fig. 1. The present state is available at the output of the second stage  $Y_i$ . When the next input state is 1, the next state information is specified by  $\hat{Y}_i$ . To detect a sequence between a pair of states, then the state information at  $Y_i$  and  $\hat{Y}_i$  can be decoded.

If it was desired to permit a transition to state  $S_k$  only from state  $S_i$ , then  $S_i$  can be decoded from  $Y_i$  and  $S_k$  can be decoded from  $\hat{Y}_i$ . If the next state as specified by  $\hat{Y}_i$  is  $S_k$  and the present state is not  $S_i$  as specified by  $Y_i$ , then an error condition can be signaled. This is depicted in Fig. 3. To prevent the circuit from assuming state  $S_k$  under the error condition, the error signal can be fed into the NOR gate which drives transistor T in Fig. 1. The error signal would prevent the circuit transition to state  $S_k$ . Moreover, since transistor T is not enabled when the error condition is detected, the circuit will not transition to  $S_k$  and remain in  $S_i$ . It might be desirable to stop all processing when the error condition is detected. If so, the error signal can be used to disable all further input state changes and the circuit would remain in the current state without any further state transitions.  $\hat{Y}_i$  will specify the incorrect state  $S_j$ ,  $S_j \neq S_i$ . If one desired to know the value of  $S_j$ ,  $\hat{Y}_i$  could be examined to reveal the error state to help with diagnostics.

## 6 Fault Detection

Classical fault detection of sequential faults includes using an error detection code on the state assignment [8]. If hardware is not shared, a single error detection code is sufficient. Since the design approach used here does not share logic, except for the NOR gate which drives the T transistors, a single error detection code can be employed and is used in the work presented here. It is assumed that the NOR gate is hard core for this discussion. Moreover, it is assumed that only one device can fail at a time and that the circuit will assume all total circuit states before a second fault can occur. In this discussion, all faults



that can cause a false next state value are detectable; this includes stuck-at, stuck-open and stuck-on faults.

The circuit presented thus far has some interesting fault detection capabilities. Most other fault detection mechanisms for sequential circuits detect the presence of a fault after the circuit has assumed a faulty state. This circuit is able to detect the presence of a fault in most of the circuit before the circuit actually enters the fault state.

In this discussion, it is assumed that a simple parity code is used for fault detection. Under the single fault assumptions above, only one extra state variable needs to be added. Let the states of the flow table be encoded with an even parity state assignment. Whenever odd parity is assumed by the state variables, a fault condition is detectable. Let all odd parity states (fault states) be assigned to have a next state value that is also odd parity. Therefore, whenever an odd parity state is assumed, the next state is also an odd parity state.

The circuit for fault detection is shown in Fig. 4. The fault detector simply detects the presence of odd parity on the state assignment;  $f$  is assigned to equal 1 when an odd parity state is present. The fault detector monitors the parity of  $\hat{Y}_i$ . If a fault occurs to any of the circuitry that produces  $\hat{Y}_i$ ,  $f$  will detect its presence. With a fault,  $f = 1$ , and since  $f$  feeds into the NOR gate, the T transistor is not enabled and the fault state cannot be assumed by  $Y_i$ . In this case, the circuit does not enter the fault state. Moreover, if the input states can be disabled, the circuit will remain in the current state.

Signal  $f$  will be driven towards a 1 value as the circuit transitions between unstable and stable states. Signal  $f$  then would prevent the T transistor from being enabled, but this actually helps the circuit not enter an improper state. Signal  $f$  can be used therefore to produce a self synchronizing signal, but this is a subject beyond the scope of this paper.

If a fault occurs in the second stage after the T transistor, then an odd parity state will be entered. The next state value as specified by the  $f_{ip}$  terms will be odd parity also since it is assumed that only one fault is present. If the  $f_{ip}$  terms generate odd parity, then  $\hat{Y}_i$  will also have odd parity and then  $f = 1$  with the fault being detected.

A fault in a T transistor will have the same impact as a fault in the second stage. If  $Y_i$  assumes the correct value in spite of a faulty T, no error is detected and the circuit operates as designed. Only when  $Y_i$  assumes an incorrect value will an odd parity state be entered and hence detected.

## 7 Summary

A fundamental logic circuit has been presented that will allow for efficient implementation of pulse mode asynchronous sequential circuits. Level input flow tables can be transformed into transition sensitive flow tables which can be directly implemented with the circuit presented here. The resulting circuits are tolerant of 1-1 crossover conditions. The final next state of the circuit is determined by the last input that is 1 whenever more than one input state is 1.

The unique characteristic of state sequence detection can be achieved with this circuit.

It is very easy to detect the present and next state in the circuitry and to prevent next state transitions to occur. In addition to state sequence detection, real time fault detection can be achieved where a fault state can be detected prior to a transition to a fault state. This fault detection capability covers a wide range of fault conditions and possible faults in the circuit.

## References

- [1] K. Cameron, S. Whitaker and J. Canaris, "ACE: Automatic Centroid Extractor for Real Time Target Tracking", NASA Symposium on VLSI Design, pp. 8.2.1-8.2.8, Nov, 1991.
- [2] J. Bredeson and P. Hulina, "Synthesis of Multiple-Input Change Asynchronous Circuits Using Transition-Sensitive Flip-Flops", IEEE Trans. Comput., vol. C-32, no. 5, pp. 37-44, May 1973.
- [3] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp 238-242.
- [4] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78
- [5] J. Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, Aug. 1966, pp. 551-560.
- [6] M. Liu, K. Liu, G. Maki and S. Whitaker, "Automated ILA Design for Synchronous Sequential Circuits", NASA Symposium on VLSI Design, Vol 3, October 1991.
- [7] J. Smith and C. Roth, "Analysis and Synthesis of Asynchronous Networks Using Edge Sensitive Flip-Flops", IEEE Trans on Computers, vol. C-20, pp. 847-855, Aug 1971.
- [8] John Meyer, "Fault Tolerant Sequential Machines", IEEE Transactions on Computers, vol. C-20, October 1971. sequential circuits. IEEE TC around 1970.

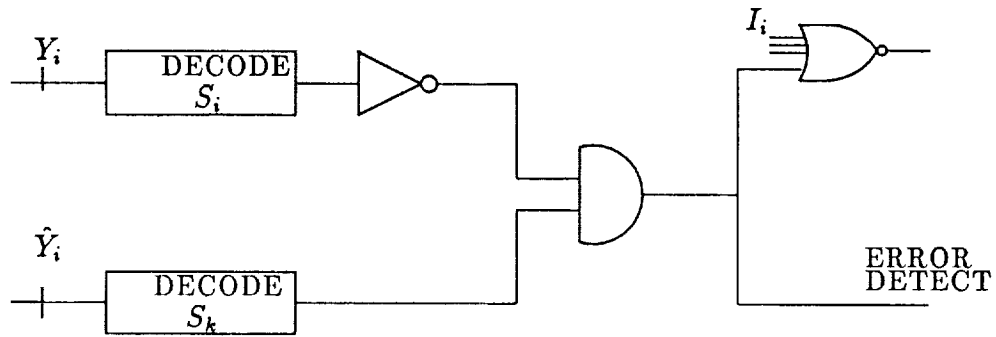


Figure 3: State Sequence Detection Logic

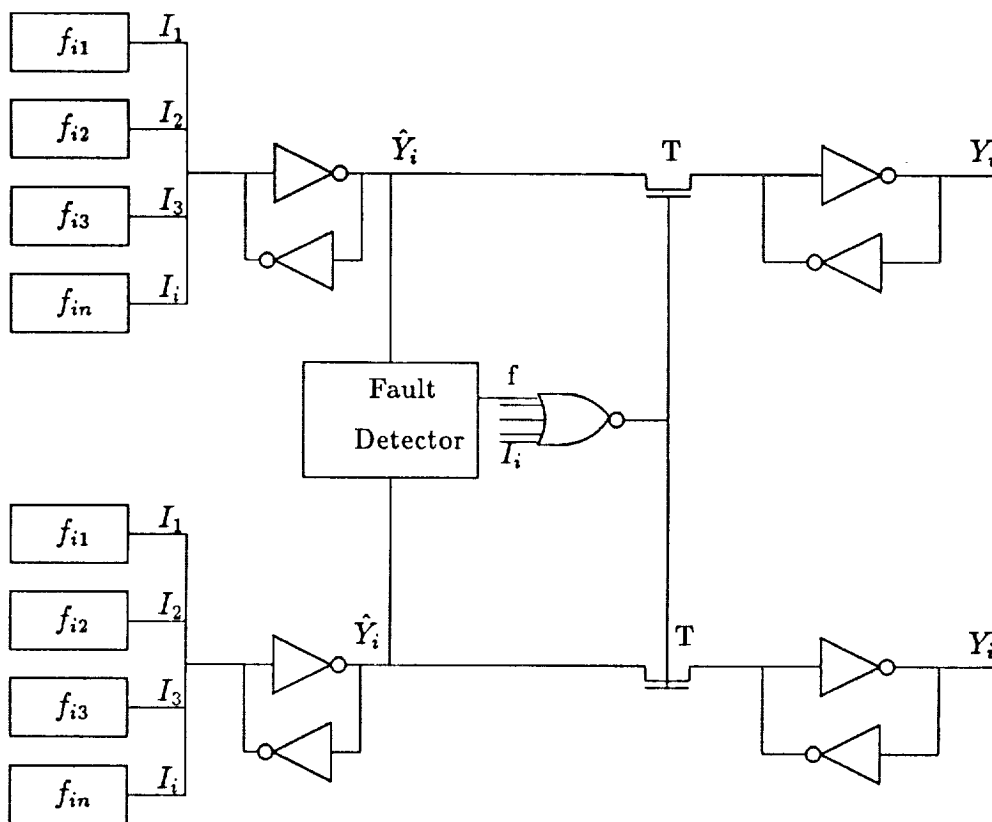


Figure 4: Fault Detection Logic



# Improved Self Arbitrated VLSI Asynchronous Circuits

P. Winterrowd

NASA Space Engineering Research

Center for VLSI System Design

University of Idaho

Moscow, Idaho 83843

**Abstract-** This paper introduces an improved method for designing the class of CMOS VLSI asynchronous sequential circuits introduced in the paper by Sterling R. Whitaker and Gary K. Maki, "Self Arbitrated VLSI Asynchronous Circuits."

## 1 Introduction

Synchronous sequential circuits are often the first choice in VLSI design. Races are avoided by synchronizing the circuit with a common clock signal; however, the frequency of this signal must be slow enough to allow signals to propagate through the slowest block regardless of how often that block's output is actually used. Also, the RC delays introduced in VLSI design make synchronizing the circuit with a clock signal increasingly difficult as the circuit's complexity increases. Moreover, with CMOS circuits peak power usage is attained during switching. If several blocks are synchronized by a clock signal and, thus, switching at the same time then the peak power required by the chip is greatly increased.

These limitations can be avoided by designing with asynchronous circuits. The paper by S. Whitaker, "Self Arbitrated VLSI Asynchronous Circuits", presents an asynchronous circuit with some interesting qualities. Of main interest here is the simple design by inspection rules that arise from this circuit. This paper presents a variation on Whitaker's circuit which reduces the number of transistors required.

## 2 Circuit Model

The general model for this circuit is the same as that given in Whitaker's research. There is an enable and disable block feeding into a buffer stage as shown in Figure 1.

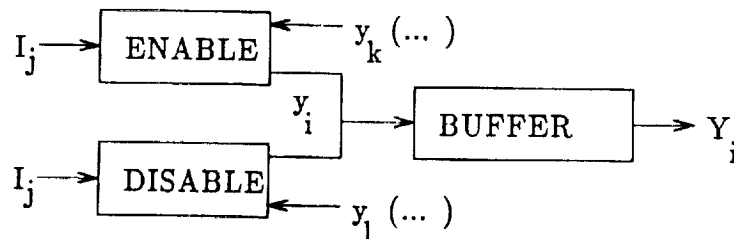


Figure 1: General Circuit Model

Where  $y_i$  and  $Y_i$  are present and next state variables respectively, and  $I_i$  represents input signals.

The variation on the original circuit presented here differs in how the buffer, enable, and disable blocks are implemented. Whitaker's buffer circuit consisted of two inverters and two weak feedback transistors as shown in Figure 2:

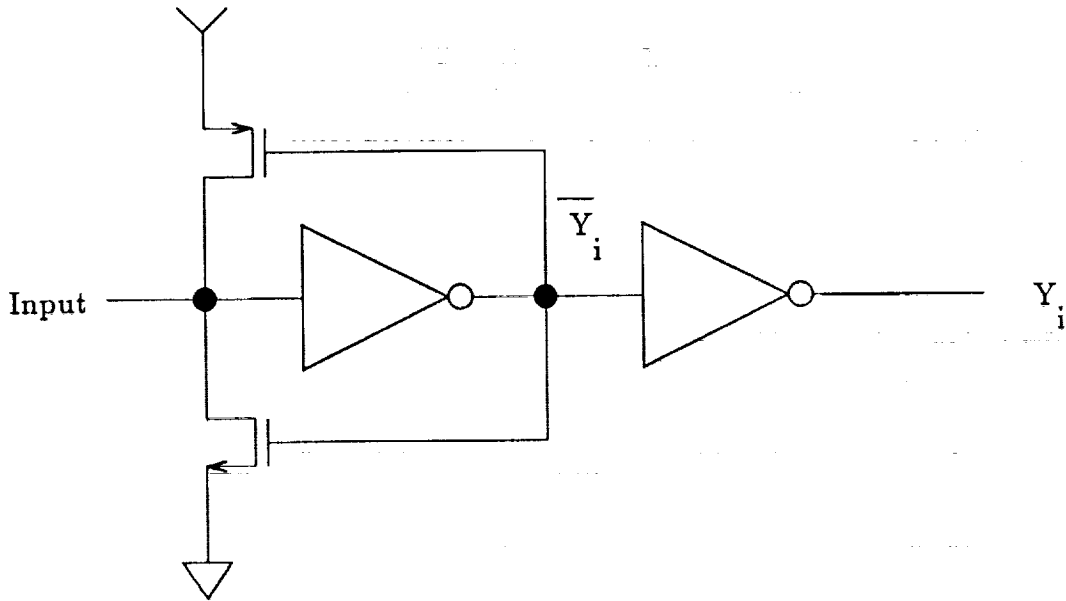


Figure 2: Original Buffer Circuit

As shown, this buffer circuit provides not only  $Y_i$  but also  $\overline{Y_i}$ . The buffer state table for this circuit is given in table one.

Table 1: Original Buffer State Table

$y_i$	Input	$Y_i$
0	0	0
0	1	1
1	0	0
1	1	1
0	Z	0
1	Z	1

The buffer for the circuit presented in this paper is shown in Figure 3. Although it only produces the  $Y_i$  variable and not its complement, it saves two transistors and in the design procedure for this circuit the complemented variable is not needed. The buffer state table for this circuit is given in Table 2.

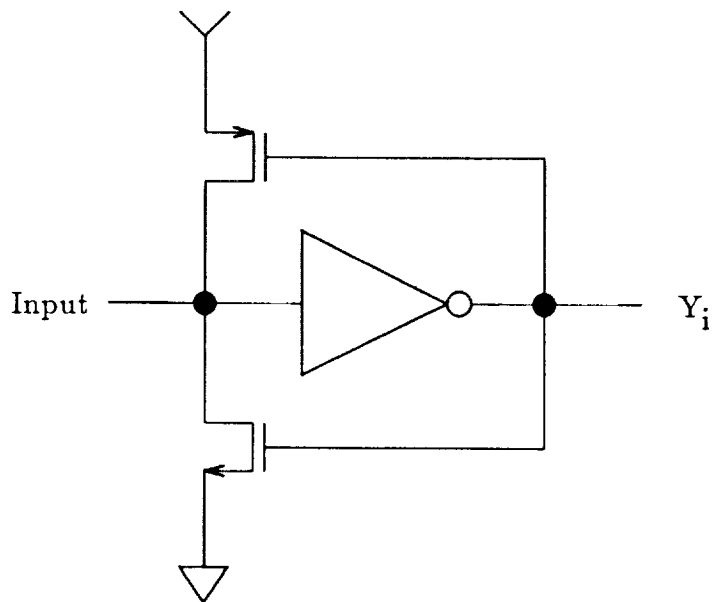


Figure 3: New Buffer Circuit

The enable and disable blocks are simple pass networks and their design is completely specified by the design equations given later in this paper.

Table 2: New Buffer State Table

$y_i$	Input	$Y_i$
0	0	1
0	1	0
1	0	1
1	1	0
0	Z	0
1	Z	1

### 3 State Assignment

The state assignment for a flow table remains the same in this paper as in its predecessor, a simple one-hot-code state assignment as shown in Table 3.

Table 3: Example Flow Table

	$I_1$	$I_2$	$I_3$	$y$	$y$	$y$	$y$	$y$	$y$
				a	b	c	d	e	f
A	(A)	E	F	1	0	0	0	0	0
B	A	(B)	F	0	1	0	0	0	0
C	A	B	(C)	0	0	1	0	0	0
D	(D)	B	C	0	0	0	1	0	0
E	D	(E)	C	0	0	0	0	1	0
F	D	E	(F)	0	0	0	0	0	1

These circuits effect a non-normal transition. Thus, in order to show that the circuit operates correctly, it is necessary to show that no transition path between two states overlaps any transition path between two other states.

**Definition 1:** Let  $[y_i]$  be the state with bit  $y_i$  set and let  $[y_a y_b y_c \dots]$  be the binary number with the appropriate number of bits with the  $y_a, y_b, y_c \dots$  bits set.

**Definition 2:** By definition (to be shown later) let a proper transition path between  $[y_i]$  and  $[y_j]$  be  $[y_i y_j]$ . Note that from the one hot code state assignment every state can be expressed by  $[y_k]$ , where  $y_k$  represents the one bit which should be set for any state.

**Theorem 1:** Given a one-hot-code state assignment with the above "proper" transition path, no two transition paths will overlap.

**Proof:** Since the transition path for two states  $[y_i]$  and  $[y_j]$  is given by  $[y_i y_j]$  this will overlap with the transition  $[y_a]$  and  $[y_b]$  given by  $[y_a y_b]$  only if  $y_a = y_i$  and  $y_b = y_j$ ; thus, the transition between two states only overlaps itself.

Therefore, to show that the circuit presented here correctly implements a flow table it must simply be shown that it correctly implements the "proper" transition path referred to above.

## 4 Design Procedure

This section starts with a definition.

**Definition 3:** Let a scale-of-two of loop in a flow table be defined by a any state  $A$  under input  $I_k$  which goes to state  $B$  under input  $I_j$  where state  $B$  returns to state  $A$  under input  $I_k$ .

The basic design equations for a circuit can be expressed as:

$$\Sigma(I_j \Sigma y_k(0)) + \Sigma y_i(1) \quad (1)$$

Enable Expr. + Disable Expr.



Note that the disable expression form only holds in the absence of order-of-two loops.

Looking at an example flow table as given in Table 3 and reproduced here for convenience:

Table 3: Example Flow Table

	$I_1$	$I_2$	$I_3$	$y$	$y$	$y$	$y$	$y$	$y$
				a	b	c	d	e	f
A	$\textcircled{A}$	E	F	1	0	0	0	0	0
B	A	$\textcircled{B}$	F	0	1	0	0	0	0
C	A	B	$\textcircled{C}$	0	0	1	0	0	0
D	$\textcircled{D}$	B	C	0	0	0	1	0	0
E	D	$\textcircled{E}$	C	0	0	0	0	1	0
F	D	E	$\textcircled{F}$	0	0	0	0	0	1

For each state the design procedure is simple:

For state  $[y_i]$ :

1. Identify all input states  $I_j$  under which  $[y_i]$  is stable. These input states become the  $I_j$ 's in the basic equation.
2. For each  $I_j$  identify all unstable states  $[y_i]$  in the same column and note the stable state's row  $[y_k]$  under which they occur. Again, these  $[y_k]$ 's become the  $y_k$ 's in the basic equation under the appropriate  $I_j$ 's.
3. Thus, the enable equation can be written as:

$$\Sigma(I_j \Sigma y_k(0)) \quad (2)$$

4. Identify all unstable states  $[y_l]$  under the row for the stable state  $[y_i]$ .
5. The disable expressions for the state  $[y_i]$  can be written as:

$$\Sigma y_l(1) \quad (3)$$

6. For each term in the disable expression determine if it is a member of a scale-of-two loop. If so, include the input state under which this term occurs as part of the term. For example, if  $y_l(1)$  was under  $I_k$  and was a member of a scale-of-two loop, then  $y_l(1)$  would become  $I_k y_l(1)$ .

As an example, the enable expression for state A,  $[y_a]$ , would be:

$$I_1(y_b(0) + y_c(0)) \quad (4)$$

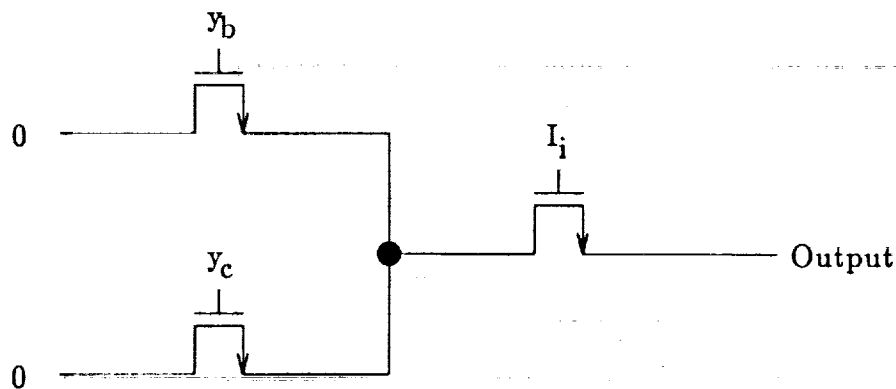


Figure 4: Example Enable Block

And, the disable expression for state A would be:

$$y_e(1) + y_f(1) \quad (5)$$

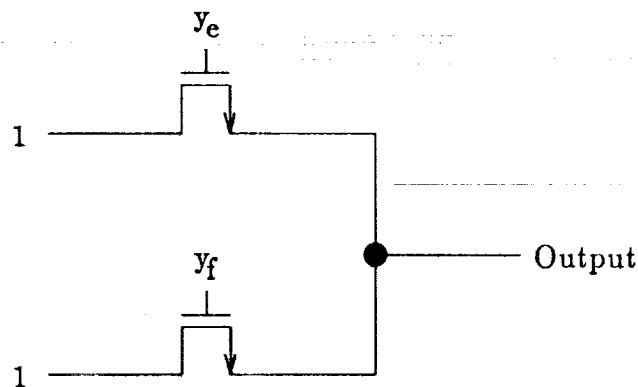


Figure 5: Example Disable Block

## 5 Circuit Operation

In order to show that this circuit operates correctly, it must be shown that the circuit transitions properly. To do this it must be shown that in going from state  $[y_i]$  to  $[y_j]$  under input  $I_j$  the transition path is  $[y_i, y_j]$ .

**Theorem 2:** *Only positive and not complemented variables are used in the design equations for this circuit.*

**Proof:** This follows directly from the design procedure and the basic design equation.

**Theorem 3:** *The enable and disable equations for  $[y_i]$  do not contain the present state term  $y_i$ .*

**Proof:** The design equations for state  $[y_i]$  derive their terms from the unstable states in row  $[y_i]$  and the stable states in the other rows which contain unstable  $[y_i]$  states. Since every state  $[y_i]$  in the row for the state  $[y_i]$  is stable and every state  $[y_i]$  not in the row for state  $[y_i]$  is unstable, then the  $y_i$  term never appears in the design equations for state  $[y_i]$ .

**Theorem 4:** *For a reduced row flow table, for each unstable state  $[y_u]$  under input  $I_u$  on the stable state  $[y_s]$ 's row that unstable state will contribute only the following terms to the design equations for that table:*

$$\begin{aligned} &y_u(1) \text{ or } I_u y_u(1) \text{ disable term for } Y_i \\ &I_u y_s(0) \text{ enable term for } Y_u \end{aligned}$$

**Proof:** The proof follows directly from the design procedure.

**Theorem 5:** *For a reduced row flow table that transitions from state  $[y_i]$  to the state  $[y_k]$  under input  $I_j$ , then while the circuit is in state  $[y_i]$  under input  $I_j$  the only variable to be affected is  $Y_k$  which is set high.*

**Proof:** If this theorem were not true then either 1)  $Y_i$  would be set to 0 from this state, or 2)  $Y_k$  would not be changed, or 3) Some other variable (all of which are zero at this point due to one-hot-code state assignment) would be raised to a 1.

First, for  $Y_i$  to be set to zero with all other state variables at zero then the design equation would have to contain the term  $y_i(1)$ , which is invalid from Theorem 3, or  $\overline{y_k}(1)$ , which is invalid from Theorem 2.

Second,  $Y_k$  will be set high from Theorem 4 and the design procedure which state that the enable expression for  $Y_k$  will include  $I_j y_k(0)$ . Note that a conflict could occur if the design equation for  $Y_k$  contained  $y_i(1)$ ; however, this would be a scale-of-two loop and the  $y_i(1)$  term would be  $I_k y_i(1)$  where  $I_k$  is different from  $I_j$ . Thus, there would be no conflict for a flow table properly designed and  $Y_k$  will be set high.

Finally, for another variable  $[y_m]$  to be set high from the state  $[y_i]$  under  $I_j$  then it's enable equation would have to contain the term  $I_j y_i(0)$  which would, from the design procedure, mean that under the row for state  $[y_i]$  under  $I_j$  would be state  $[y_m]$ ; however, by definition this location contains  $[y_k]$ .

Thus, the theorem must be true since all other alternatives are false.

**Theorem 6:** *If the circuit is in the state  $[y_i y_k]$  under input  $I_j$  where state  $[y_i]$  goes to state  $[y_k]$  under input  $I_j$  then the only variable to be affected is  $Y_i$  which is set low.*

**Proof:** If this theorem were not true then either 1)  $Y_i$  would stay high, or 2)  $Y_k$  would be set low, or 3) Some other variable would be set high (since all the other variables are, by definition, low to begin with.)

First, from Theorem 4 and the design procedure the design equations for  $Y_i$  contains the term  $y_k(1)$  or  $I_j y_k(1)$ ; thus, it would remain only not go low if it also included  $I_j y_i(0)$ , which is invalidated by Theorem 3, or  $I_j \overline{y_i}(0)$ , which is invalidated by Theorem 2, or  $I_j y_k(0)$  which can be invalidated by the following argument. If the enable expression for

$Y_i$  contained any  $I_j$  terms then the position under  $I_j$  would have to contain  $[y_i]$  and, by definition, it contains  $[y_k]$ . Thus,  $Y_i$  is set low.

Second, for  $Y_k$  to be set low it would contain the term  $y_k(1)$  or  $I_j y_k(1)$ , which can be invalidated from theorem three, or  $\bar{y}_i(1)$ , which can be invalidated by theorem two, or  $y_i(1)$  which can be invalidated by the following argument. If  $Y_k$  did contain  $y_i(1)$  then that would be a scale-of-two loop and the  $y_i(1)$  term would have to be  $I_m y_i(1)$  where  $I_m$  is not equal to  $I_j$ ; thus,  $Y_k$  will not be set low.

Finally, for another variable  $[y_m]$  to be set high from the state  $[y_i, y_j]$  under  $I_j$ , the enable equation would have to contain either the term  $I_j y_i(0)$  or  $I_j y_j(0)$ . The term  $I_j y_i(0)$  would, from the design procedure, mean that under the row for state  $[y_i]$  under  $I_j$  would be state  $[y_m]$  which by definition contains  $[y_k]$ . The term  $I_j y_j(0)$  would, from the design procedure, mean that under the row for state  $[y_j]$  under  $I_j$  would be state  $[y_m]$  which by definition contains the stable state  $[y_j]$ . Thus, no other variable will be set high.

Therefore, since all the other alternatives are proven false, this theorem must be true.

So from Theorems 5 and 6 the circuit created from the design procedure in the previous section fulfills Theorem 1 and is critical-race free.

## 6 Transistor Count Comparison

From the assignment procedure it is obvious that we need one state variable from each state. Also, from Figure 1, which shows the general circuit model for this circuit, it is clear that for each state variable we need four transistors (one buffer.) Moreover, from Theorem 4 and the design procedure it can be shown that for each stable state  $[y_i]$  in a flow table one transistor is required if an unstable state  $[y_i]$  is also in that column and for every unstable state two transistors are required. Also, every scale-of-two loop contributes an additional two transistors. Thus, the number of transistors needed for a reduced row flow table is given by:

$$\text{Total \# of Transistors} = 4S + B + 2U + 2L \quad (6)$$

Where  $s$  = number of states,  $b$  = number of stable states in the low table with identical unstable states in the same column,  $u$  = number of unstable states in flow table, and  $L$  = number of scale-of-two loops that exist in the flow table. The number of transistors for the design method in Whitaker's paper can be shown to be:

$$\text{Total \# of Transistors} = 6S + 4U + 2L \quad (7)$$

Thus, for the example flow table given in Table 3 where  $s=6$ ,  $b=6$ ,  $u=12$ , and  $L=0$  the total number of transistors for the improved design method is 54, and the total number for the old method is 84; thus, a difference of 30 transistors.

Table 4 shows some typical values for reduced row flow tables and compares transistor counts.

Table 4: Transistor Count Comparison

S	B	U	L	Transistor Count		New/Old
				Old Method	New Method	
6	6	12	0	84	54	0.643
8	9	15	1	110	73	0.664
7	10	18	2	118	78	0.661
9	10	17	0	122	80	0.656

## Reset Feature

In any circuit, asynchronous or synchronous, it is advantageous to be able to preset the circuit to some state. This is almost a necessity upon startup since a circuit often begins in an unknown or undesirable state. The basic model for this circuit can be modified to easily include a reset feature as shown in figure six:

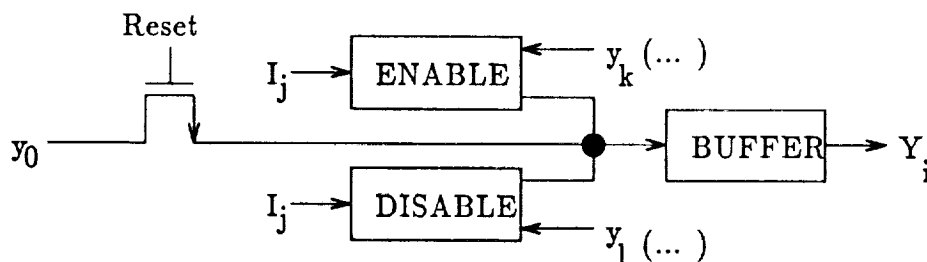


Figure 6: Modified General Circuit Model

This feature only costs one transistor for each state; however, the design must insure that all inputs are low while the reset is high.

## Summary

The general circuit model for this paper and the one-hot-code state assignment lead to an easily designed and implemented asynchronous circuit. Once an input is introduced the circuit sets the new variable high which then sets the variable signifying the old state low. This  $[y_i]; [y_i y_j]; [y_j]$  non-normal mode transition insures that no two transition paths overlap; thus, the circuit is critical-race free.

This improved design method reduces the transistor count from the old method by, roughly, one third, decreasing the size of the overall circuit and increasing its usefulness.

Finally, although the circuit operates at 1/2 the speed of an STT state assignment asynchronous circuit due to its non-normal mode operation it is very easy to design and avoids the disadvantages of a synchronous circuit such as clock routing, power bussing, and speed dependency upon slowest information path.

## References

- [1] S. Whitaker, S. Manjunath and G. Maki, Self Arbitrated VLSI Asynchronous Circuits, *NASA SERC 1990 Symposium on VLSI Design*, pp. 87-103.
- [2] S. Golpalakrishnan, G. Kim and G. Maki, Implications of Tracey's Theorem to Asynchronous Sequential Circuit Design, *NASA Symposium on VLSI Design*, pp. 9.1.1-9.1.11, Nov. 1990.
- [3] S. Whitaker and G. Maki, Pass Transistor Asynchronous Sequential Circuits, *IEEE Journal of Solid State Circuits*, pp. 71-78, Feb 1989.

This research was supported (or partially supported) by NASA under Space Engineering Research Center Grant NAGW-1406.

## **A Special Purpose Silicon Compiler For Designing Supercomputing VLSI Systems**

\* N.Venkateswaran, P.Murugavel, V.Kamakoti, M.J.ShankarRaman, S.Rangarajan,  
M.Mallikarjun, B.Karthikeyan, T.S.Prabhakar, V.Satish, P.R.Venkatasubramaniam,  
R.Sivakumar, R.Srinivasan, S.Chandrasekhar, G.Suresh, M.B.Karthikeyan,  
S.Ramachandran, S.Sankar, P.V.Balaji, P.Kishore

\*\* F.Lawrence, S.Pattabiraman, G.Suresh, V.Arun Shankar, A.Ashraf, V.Balaji,  
M.Balaji, Sunil.K, R.Devanathan, Y.Eliyas, K.Krishnan, B.Krishna Kumar,  
B.Kumaran, N.Rajesh & G.Vijay Venkatesh

\*\*Department of Computer Science and Engineering.  
Sri Venkateswara College of Engineering  
University of Madras  
Nazarethpet, Madras 602 103, India.

**Abstract-** Design of general/special purpose Supercomputing VLSI systems for numeric algorithm execution involves tackling two important aspects namely their computational and communication complexities. Development of software tools for designing such systems itself becomes complex. Hence a novel design methodology has to be developed. For designing such complex systems a special purpose silicon compiler is needed in which

1. The computational and communicational structures of different numeric algorithms should be taken into account to simplify the silicon compiler design.
2. The approach is macrocell based.
3. The software tools at different levels, algorithm down to the VLSI circuit layout, should get integrated.

In this paper a special purpose silicon (SPS) compiler based on PACUBE macrocell VLSI ARRAYS [1] for designing supercomputing VLSI systems is presented. It is shown that turn-around-time and silicon real estate get reduced over the silicon compilers based on PLAs, SLAs and gate arrays.

Characteristics 1 and 2 above enable the SPS compiler to perform systolic mapping (at the macrocell level) of algorithms whose computational structures are of GIPOP (Generalized Inner Product Outer Product) form [2]. Direct systolic mapping on PLAs, SLAs and gate arrays is very difficult as they are micro-cell based. A novel GIPOP processor is under development using this special purpose silicon compiler.

---

\* pursuing their higher studies/employed in India or abroad. Contact for communication regarding this paper N.Venkateswaran, Additional Professor, Dept of Computer Science and Engineering, Sri Venkateswara College of Engineering, University of Madras, India. Authors' names listed randomly.

## 1 Introduction

No silicon compiler has yet been developed exclusively for tackling the complexity in designing supercomputing VLSI systems. In developing such a compiler besides achieving reduced turn-around-time and area, computational performance of mapped functional units and architectural characteristics like systolic mapping should also be considered. The latter two factors are not taken care of in micro-cell based silicon compilers.

In conventional compilers the software tools are not integrated from algorithm down to the VLSI circuit level. The integration of software tools at different levels can be achieved by adopting novel methodologies in designing supercomputing VLSI systems (processors and arrays) and developing novel algorithm mapping techniques. This integration reduces the software complexity to a great extent.

The turn-around-time is high if gate arrays, PLAs and SLAs are employed for supercomputing system synthesis. Further the silicon compilation should take place at a much higher level than at the gate or the micro-cell level for supercomputing system design.

The PACUBE macro-cell structure is a combination of PLAs, SLAs, gate arrays and standard cells. Besides storage and logic elements an important computing unit, the DRAA (Dynamically Reconfigurable Array Adder [1]) is also present (Fig 1). The presence of the DRAA reduces the turn-around-time and silicon area drastically for designing special purpose VLSI systems. The DRAA helps in achieving high performance due to its functional and architectural characteristics. If the DRAA were to be mapped on to PLAs, SLAs and gate arrays the performance will get degraded. The special purpose silicon compiler built for mapping supercomputing systems on the PACUBE arrays achieves all the above factors.

## 2 PACUBE Macrocell Array And Systolic Mapping Tools.

### 2.1 Unified GIPOP Operations On Macrocell Arrays

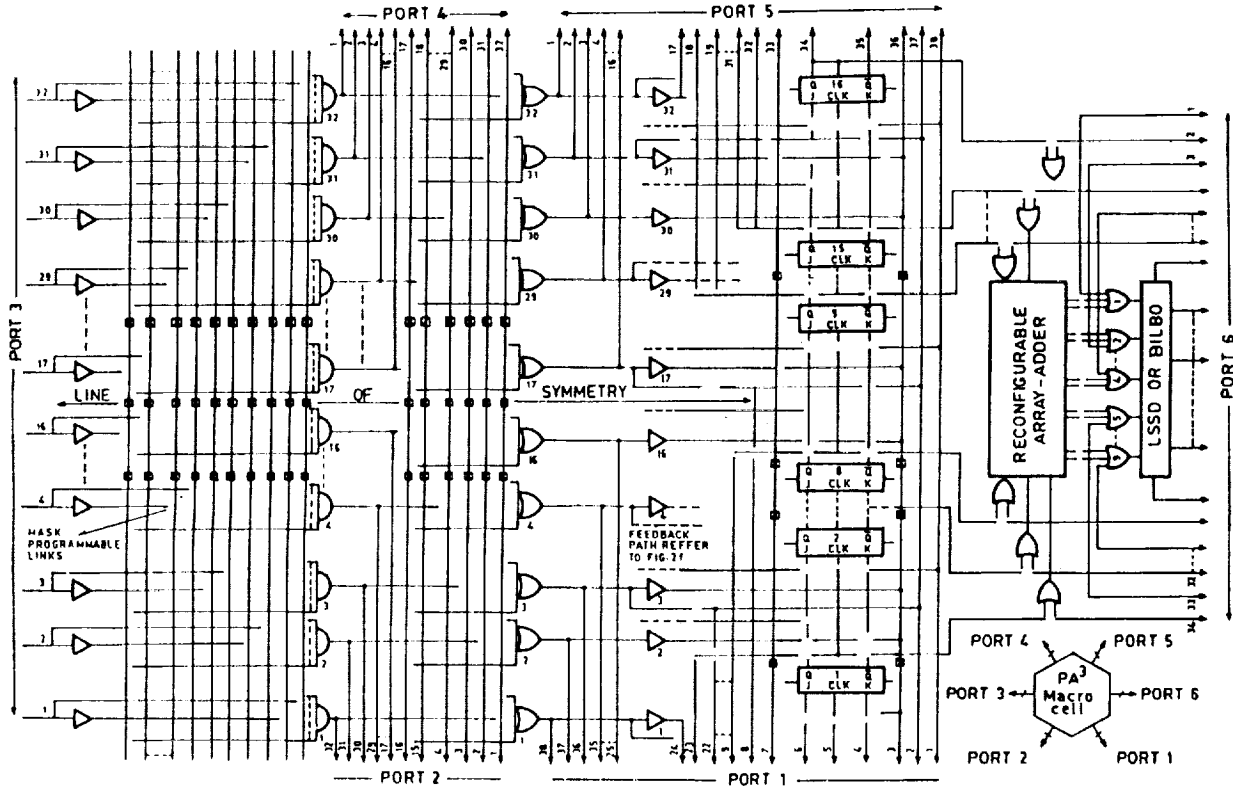
Execution of number of numeric algorithms involve inner-product operations. Several VLSI systems have been proposed for executing numeric algorithms whose computational structures are of inner-product form. For this purpose VLSI arrays of inner-product step processors have been employed conventionally.

In general the computational structure of numeric algorithms are complex. However on a closer study it is noted that these structures can be brought under a generalized form called Generalized Inner-Product Outer-Product (GIPOP) functions. These are

$$I = \sum_{i=1}^n \frac{(A_i * B_i)}{C_i} \dots \quad (1)$$

$$O = \prod_{i=1}^n \frac{(A_i + B_i)}{C_i} \dots \quad (2)$$



Figure 1:  $PAi^3$  (ASLA) Logic Model

The computational structures of numeric algorithms may involve a mathematical combination of the above two equations. Evaluation of GIPOP functions based computations involve inner-product operations, chain multiplications, outer-product operations and reciprocal operations.

It is shown in this paper that by using PACUBE VLSI arrays [1] these GIPOP functions can be evaluated as a sum of equivalent weighted inner-product functions only. Also massive parallelism can be employed in GIPOP operations. This unification of the execution processes of GIPOP functions only in terms of equivalent inner-product operations is achieved by establishing an identical inter and intra macrocell connections (data flow mapping) on PACUBE arrays for chain multiplication and inner-product operations. Both the outer-product and reciprocal operations can be expressed in terms of sum of chain multiplication operations [3].

Let the inner-product, chain multiplication, outer-product and reciprocal operations be defined as follows.

$$\sum_{i=1}^{T_i} \{w(S_i)_p * w(K_i)_p\} = I\{w(S_i)_p, w(K_i)_p, T_i\} \quad (3)$$

$$\prod_{i=1}^{T_i} \{w(S_i)_p + w(K_i)_p\} = O\{w(S_i)_p, w(K_i)_p, T_i\} \quad (4)$$

$$\prod_{i=1}^{T_i} \{w(S_i)_p\} = C\{w(S_i)_p, T_i\}$$

$$= C\{w(S_1)_p, w(S_2)_p, \dots, w(S_{T_i})_p\} \quad (5)$$

$$\frac{1}{w(S_i)_p} = R\{w(S_i)_p\} \quad (6)$$

$S_1, S_2, \dots, S_i$  and  $K_1, K_2, \dots, K_i$  are the operands of word length  $p$  bits.

$w$  - The weight of the MSB of the word length.

$T_i$  - Number of inner-product terms (operand pairs) or  
Number of chain multiplication terms (operands).

$$w(S_i)_p \rightarrow \{w_n(S_i)_{p_n}, w_{n-1}(S_i)_{p_{n-1}}, \dots, w_1(S_i)_{p_1}\} \quad (7)$$

$$w(K_i)_p \rightarrow \{w_n(K_i)_{p_n}, w_{n-1}(K_i)_{p_{n-1}}, \dots, w_1(K_i)_{p_1}\} \quad (8)$$

$p_i$  - Word length of the partition  $i$ ,  $i = 1$  to  $n$

$w_i$  - Weight of the MSB of the partition ( $p_1 + p_2 + \dots + p_i$ )

Using the relationships (7) and (8) we get

$$p = p_1 + p_2 + \dots + p_n = \sum_{i=1}^{T_i} p_i$$

$$w = w_1 + w_2 + \dots + w_n = \sum_{i=1}^{T_i} w_i \quad (9)$$

( $*$ ) Defines binary multiplication)

$$w(S_i)_p * w(K_i)_p = w+w(Q_i)_{p+p} \quad (10)$$

## 2.2 Inner-product operations on PACUBE Arrays

Execution of inner product operations on PACUBE arrays has been dealt in [1]. To achieve massive parallelism in evaluating inner-product functions partial product arrays (PPAs) of the different product terms

$$(A_1 * B_1, A_2 * B_2, \dots, A_n * B_n)$$

are obtained in parallel (forming a massive array) and added simultaneously [1]. Refer to Fig. 2a. There are three different ways of massive array formation and reduction [2]. They are called MAR1, MAR2 and MAR3 processes. The figure 2a corresponds to

$$I(3(A_i)_4, 3(B_i)_4, 4)$$

function. The reduction processes corresponding to MAR2 and MAR3 are similar to MAR1 reduction process presented in [1].

The MAR Process should be chosen such that the following important criteria are taken care of

1. The operands  $[4,4]$  sub matrices of the massive array injection into the PACUBE array should be simpler.
2. The partial sum bits output corresponding to the sum output of the massive array should occur in consecutive cycles of the array reduction process.
3. The partial sum output should be of same length in all cycles .
4. The partial sum output should occur only in the peripheral macrocells.
5. The intra macrocell data flow should be regular i.e. there is no multiplexing of data between macrocells.
6. Number of array reduction cycles and number of macrocells should be minimum.

### 2.3 Chain Multiplication On PACUBE Arrays

The application of PACUBE array can be extended for chain multiplication operation. Consider the massive array formation for

$$C\{_3(A_i)_4, 3\}$$

Similar to inner product operation three different types of massive array can be formed for executing chain multiplication. Refer to Fig. 2b. Only MAR2 massive array formation is shown.

### 2.4 Unification Of Chain Multiplication And Inner Product Operations On PACUBE Arrays

There is a striking similarity between array formation corresponding to the inner product and chain multiplication operations. The only difference is in the array sizes. The massive array of  $C\{_3(A_i)_4, 3\}$  is larger than that of  $I\{_3(A_i)_4, _3(B_i)_4, 4\}$ . In this example the difference in array sizes is not much. In general the massive arrays corresponding to the inner product and chain multiplication operations can be made of comparable sizes by a proper choice of the word length and number of terms. Hence identical array reduction process having same inter and intra macrocell data flow can be established on the PACUBE macrocell arrays. The operand injection points differ for these operations. Hence structurally these two operations are equivalent. Such equivalent pairs may not exist for certain values of word length and number of terms. In some cases even if the equivalent pairs exist the word length of the pairs may be of odd values. The word length of the equivalent pairs should have values in powers of 2. It is preferable to have equal word length for such equivalent pairs. The values of number of terms (problem size) can be adjusted to achieve this.

For example in the equivalent pair  $I\{_8(S_i)_8, _8(K_i)_8, 8\}$  and  $C\{_8(S_i)_8, 3\}$  the word lengths are same but the number of terms are different. The equivalent pair

(a)		(b)	
	$a_1b_1c_1, a_2b_1c_1, a_3b_1c_1, a_1b_2c_1, a_2b_2c_1, a_3b_2c_1, a_1b_3c_1, a_2b_3c_1, a_3b_3c_1$		$a_1b_1, a_2b_1, a_3b_1, a_1b_2, a_2b_2, a_3b_2, a_1b_3, a_2b_3, a_3b_3$
	$a_1b_1c_2, a_2b_1c_2, a_3b_1c_2, a_1b_2c_2, a_2b_2c_2, a_3b_2c_2, a_1b_3c_2, a_2b_3c_2, a_3b_3c_2$		$c_1d_1, c_2d_1, c_3d_1, c_1d_2, c_2d_2, c_3d_2, c_1d_3, c_2d_3, c_3d_3$
	$a_1b_1c_3, a_2b_1c_3, a_3b_1c_3, a_1b_2c_3, a_2b_2c_3, a_3b_2c_3, a_1b_3c_3, a_2b_3c_3, a_3b_3c_3$		$e_1f_1, e_2f_1, e_3f_1, e_1f_2, e_2f_2, e_3f_2, e_1f_3, e_2f_3, e_3f_3$
	$a_1b_1c_4, a_2b_1c_4, a_3b_1c_4, a_1b_2c_4, a_2b_2c_4, a_3b_2c_4, a_1b_3c_4, a_2b_3c_4, a_3b_3c_4$		$g_1h_1, g_2h_1, g_3h_1, g_1h_2, g_2h_2, g_3h_2, g_1h_3, g_2h_3, g_3h_3$

 $C\{3(S_1)_4, 3\}$ 
 $I\{3(S_1)_4, 3(K_1)_4, 4\}$ 

### NUMERIC ALGORITHM MAPPING CONCEPT

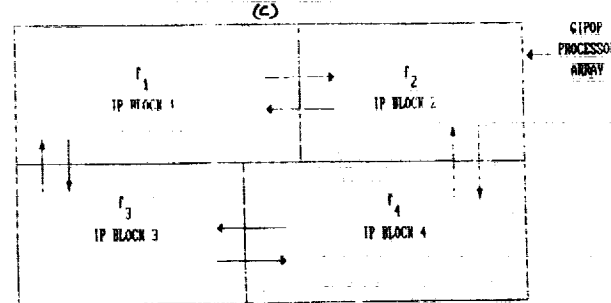


Figure 2: MAR process-2 array formation

$I\{16(S_i)_{16}, 16(K_i)_{16}, 4\}$ ,  $C\{16(S_1)_{16}, 8(S_2)_8, 8(S_3)_8, 3\}$  has different word length and problem size. But  $C\{16(S_1)_{16}, 16(S_2)_{16}, 16(S_3)_{16}, 3\}$  can be easily decomposed (by proper word length and term/problem size partitioning) in terms of  $C\{16(S_1)_{16}, 8(S_2)_8, 8(S_3)_8, 3\}$ . That is  $C\{16(S_i)_{16}, T_i\}$  can be decomposed to  $I\{16(S_i)_{16}, 16(K_i)_{16}, 4\}$ . Further details on this is dealt in section 3.3.

This leads to a unified PACUBE VLSI array for executing Inner product and Chain multiplication operations. Outer product operation and high speed multiplicative division algorithm [3] are based on chain multiplication operation. Hence the execution processes of GIPOP functions can be unified on the PACUBE macrocell arrays.

## 2.5 Systolic Mapping Of GIPOP Functions

An algorithm has been developed for automatic systolic mapping of GIPOP function execution on the P-arrays and implemented under DOS. (Fig 3). The unification of the execution processes of the GIPOP functions has greatly simplified the development of systolic mapping tools.

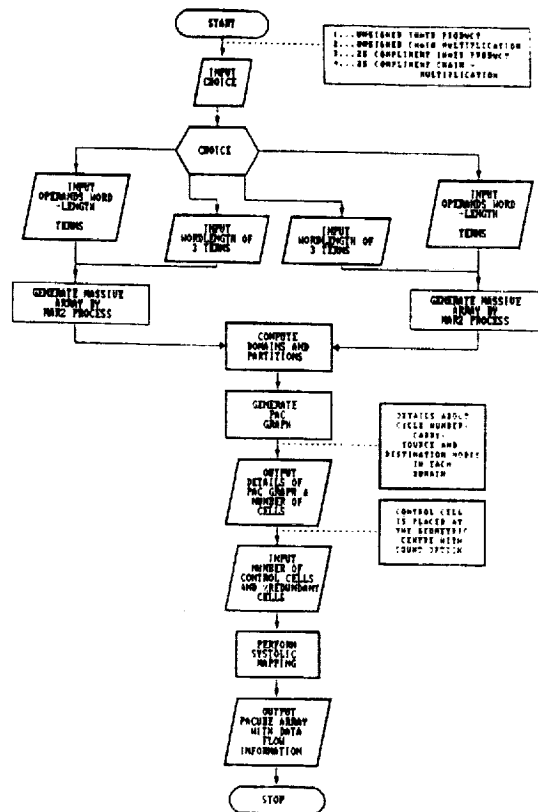


Figure 3: Systolic mapping of GIPOP functions on Pacube arrays

### 3 GIPOP Processor Array And Software Tools For Algorithm Mapping

#### 3.1 GIPOP Processor

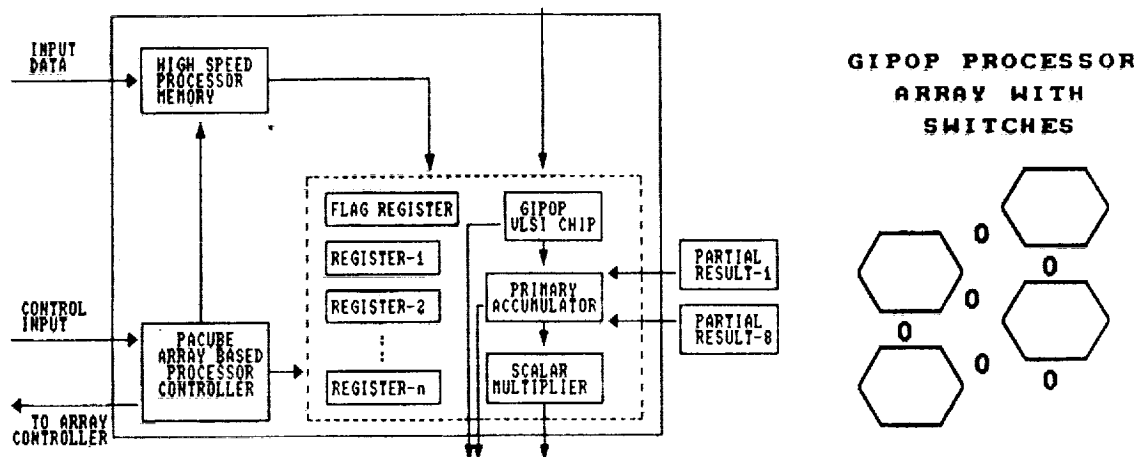
Existing processor arrays meant for supercomputing are classified into special purpose and programmable general purpose arrays. An attempt to combine the advantages of these approaches has culminated in a novel processor design approach, namely GIPOP processor arrays. This novel approach is expected to give very high performance/cost ratio for supercomputing systems [2].

The internal architecture of the GIPOP processor and its instruction set is shown in Figure 4. The simulation of the instruction set has been completed. The GIPOP processor can execute the equivalent pair  $I\{3(S_i)_4, 3(K_i)_4, 4\}$  and  $C\{3(S_i)_4, 3\}$ .

The equivalent pairs are chosen based on the PACUBE macrocell, GIPOP processor and the array complexities.

#### 3.2 GIPOP Processor Array

Two levels of pipelining take place in algorithm execution on the GIPOP processor array partly shown in Figure 4, one at the macro-cell level within the GIPOP chip and the other at the processor level. The Processor level pipelining is controlled by the Array Control



GIPOP INSTRUCTION SET		
INSTRUCTION	FORMAT	
MOV	DESTINATION, SOURCE, VECTOR CYCLES	
INSTRUCTION	OPERANDS	COMMENTS
MOV	GPR, M	Pipelining of data from memory
MOV	GPR, I	or input port through general purpose registers.
MOV	CM, M	Loads three four-bit operands
MOV	CM, I	for chain multiplication operation either from memory or input port.
MOV	IP, M	Loads eight four-bit inner-product
MOV	IP, I	operands from memory or input port.
MOV	PAC, M	Loads eight four-bit massive array operands
MOV	PAC, I	in to the primary accumulator.
MOV	PAC, GPOP	Accumulate the output of GIPOP in the primary accumulator.
MOV	MLT, M	Loads four-bit multiplier from memory to scalar multiplier.
MOV	MLT, PAC	Loads the multiplicand from the accumulator to scalar multiplier.
MOV	0, GPR	Transfer the output of general purpose registers,
MOV	0, GPOP	GIPOP, primary accumulator, and the scalar multiplier,
MOV	0, PAC	to the output port. (these loadings can be done in parallel).
MOV	0, MLT	

$$[I_3(Si)_{4,3}(Ki)_{4,3}C_3(S1)_{4,3}(S2)_{4,3}(S3)_{4,3}]$$

Figure 4: GIPOP processor architecture for equivalent pair

Unit (ACU) and the macrocell level pipelining is controlled by the chip control unit (CCU). The hardware complexity of the switch lattice depends on the processor complexity, data communication complexity in an algorithm and the word length. The design of the ACU is under progress.

### 3.3 Mapping Of Numeric Algorithms On GIPOP Processor Array

Mapping of numeric algorithms on the GIPOP processor array involves tackling the computational (levels 1 & 2) and communicational (level 3) complexities.

- Level 1.** Decomposing the computational structure of the algorithm in terms of GIPOP equations which are further decomposed in terms of the inner product functions only [1].
- Level 2.** The architectural capabilities of a GIPOP processor is bounded by problem size and wordlength. Suitable algorithms for problem size and word length partitioning of GIPOP functions and the corresponding software tools have been developed. The algorithm is based on definitions (3) - (6) []. Refer to Figure 5.
- Level 3.**
  - a. Proper loading of input operand frames into the high speed processor memory (Block level memory loading) from the system memory.
  - b. Programming the processor and array control units.

Numeric algorithm mapping on the GIPOP arrays basically involve mapping of different inner product blocks (IP Blocks) of variable complexities (see Fig 2c). Functionally each of these IP Blocks may correspond to different GIPOP operations. The data flow between the corresponding group of GIPOP processors (making an IP block) can be syntactically described including both the functional and behavioral aspects of the GIPOP processor [2]. Mapping of an algorithm on the arrays is to get this syntactical description of the data flow.

Software tools for levels 1 and 3 are being developed. The novel concept of mapping numeric algorithms on the GIPOP processor array shown in Figure 2c greatly simplifies the development of software tools for levels 1 and 3 above.

## 4 PACUBE Logic Level

### 4.1 Inter And Intra Cell Routing.

An efficient algorithm for inter and intra macrocell routing has been developed taking into account the shortest path considerations. The software tools are shown in Figure 6 as flowcharts.

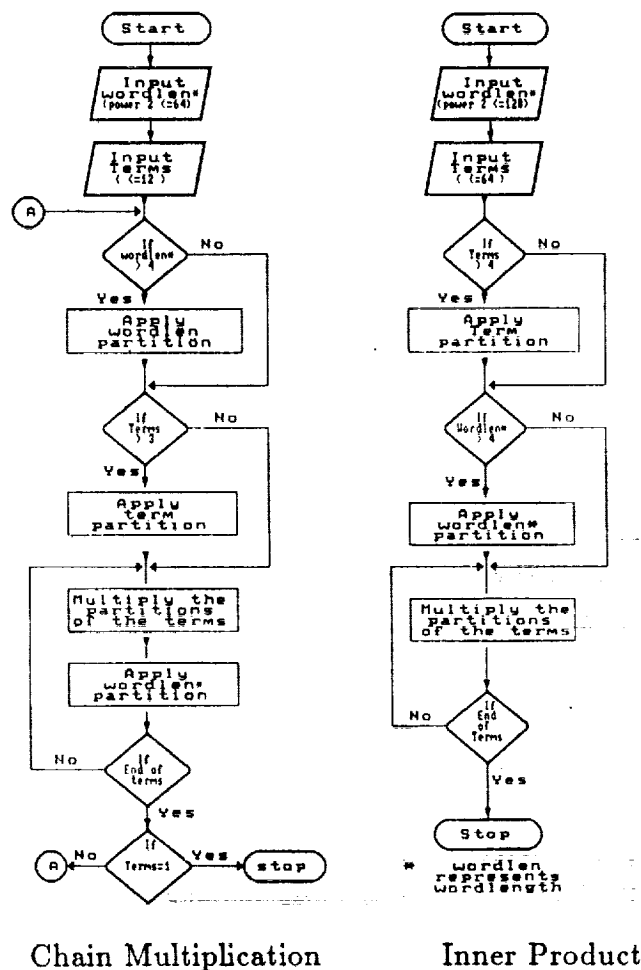


Figure 5: Wordlength &amp; term partitioning

## 4.2 Subprogram (Functional Units) Library Generation And Linking.

Several subprograms, sequential and combinatorial, have been mapped on the PACUBE macrocell. An efficient PACUBE Hardware Description Language (PHDL) has been developed. The subprogram linking is done using this PHDL and the related software tool has been developed. ( Fig. 7 )

## 5 PACUBE Circuit Level Discussions

### 5.1 Interactive Layout Generation And Checking.

A software tool for the generation of device level layouts in an interactive fashion has been developed (Fig. 8). It provides four layers viz Diffusion, Polysilicon and two levels of metal. The package supports both n-well and p-well CMOS processes. Lambda based design rules have been adopted. Special facilities such as mirroring, translation, rotation,



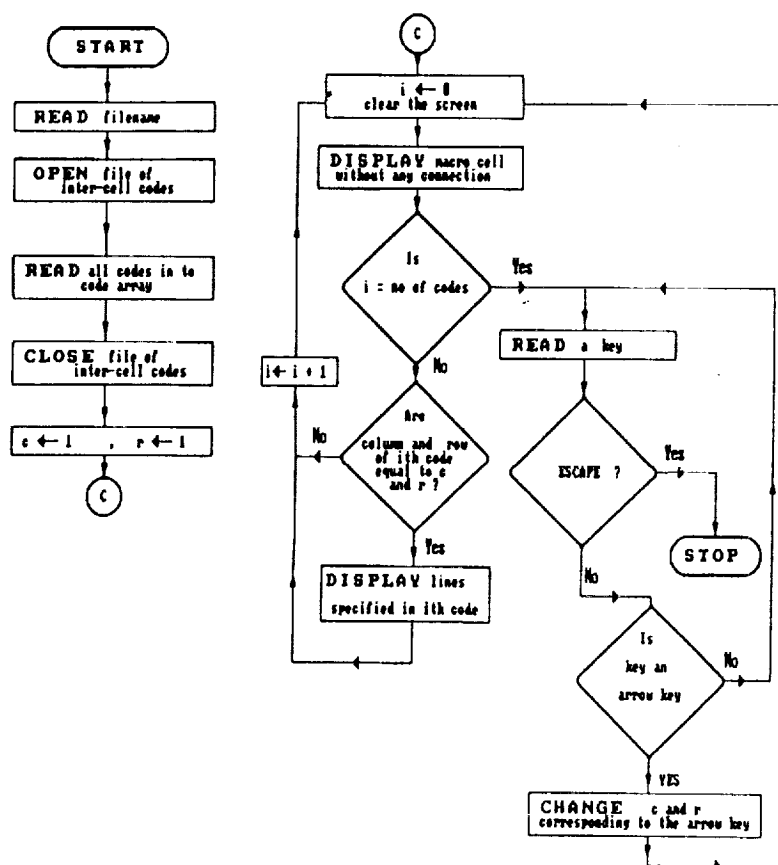


Figure 6: Inter-cell logic level tool

step & repeat, cut & paste and scaling are available to aid in faster design. The layout of an entire macrocell has been developed using this tool. Layout geometries are expressed using PACUBE device level codes. The design rule checker developed is edge based and performs both inter and intra layer checking.

The different functional units of the macrocell are treated as standard cells and depending on the application the required standard cells can be placed within the macrocell. This option gives rise to macrocell arrays with different sizes of macrocells.

## 5.2 Intracell And Intercell Mapping.

A software translator for the PACUBE logic level code to PACUBE device level code conversion and a device level to Caltech Intermediate Format (CIF) translator has also been developed.

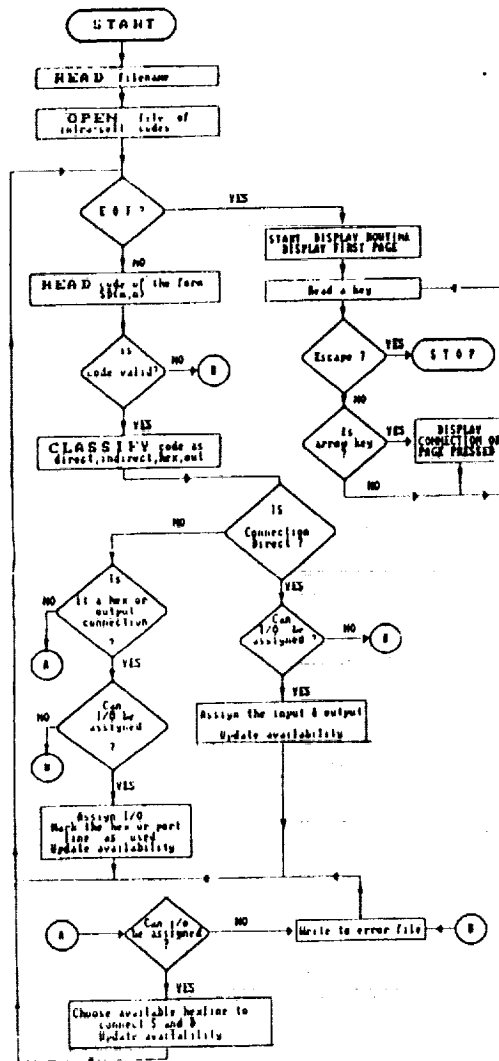


Figure 7: Intra-cell logic level tool

### 5.3 Simulation

Circuit simulation of different functional units of the macrocell for 1.2 micron technology using PSPICE is nearing completion. Logic level simulation of PACUBE macrocell for GIPOP operations has been carried out using the PACUBE logic simulator.

## 6 Conclusion

In this paper novel methodologies have been proposed for designing silicon compilers for synthesising supercomputing VLSI systems. The important criteria for such a silicon compiler are integration of its software tools and the architectural considerations.

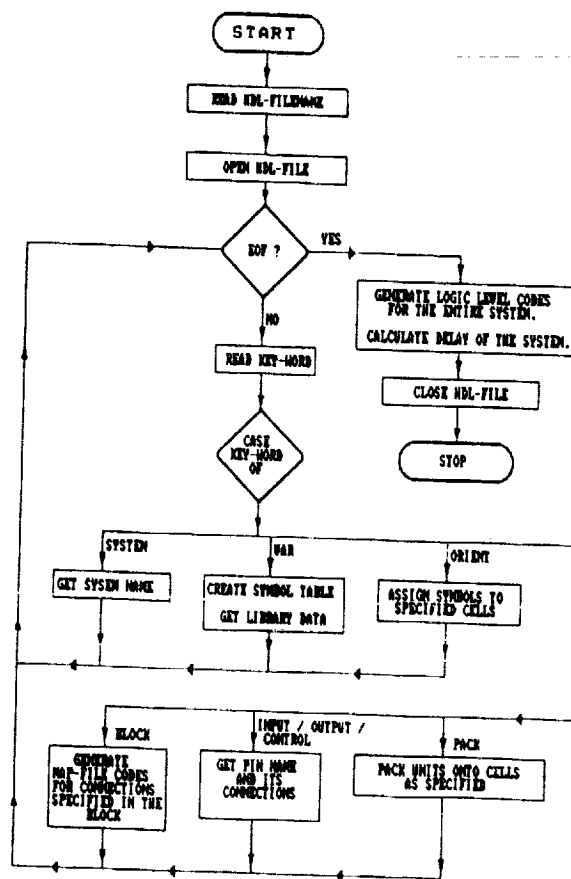


Figure 8: Sub-program linker

## 7 Future Work

An algorithm for extracting the computational complexities of numeric algorithms in terms of GIPOP functions is to be taken up. A methodology is to be developed for mapping the communication graph of numeric algorithms on to the GIPOP processor array as shown in Figure 2c. Reconfigurable fault-tolerant PACUBE arrays [2] has been developed and the corresponding software tools to incorporate this into the silicon compiler has to be taken up.

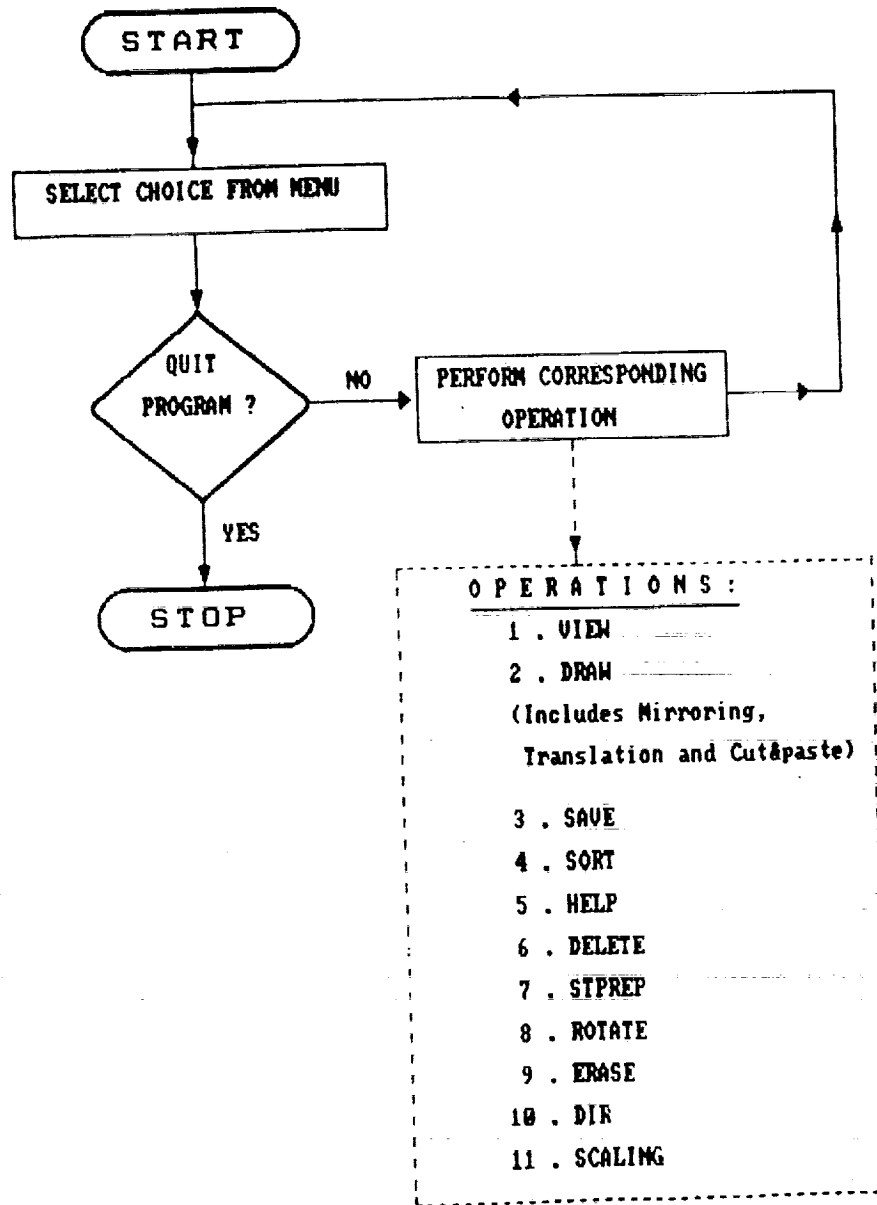
**MAIN PROGRAM :**

Figure 9: Pacube macro cell based device level layout editor

## References

- [1] N.Venkateswaran, PACUBE Arrays For Supercomputers, *Proc. of the First International Conference on Supercomputing Systems*, conducted by the IEEE Computer Society and Pentagon, held at Florida, U.S.A., Dec.6-10,1985.
- [2] N.Venkateswaran, et al., Supercomputing Systems on PACUBE VLSI Arrays, Research Report. Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, University of Madras, India.
- [3] Joseph J.F.Cavanagh, Digital Computer Arithmetic Design and Implementation, McGraw Hill publishing house.



# A Fast Adaptive Convex Hull Algorithm on Two-Dimensional Processor Arrays with a Reconfigurable BUS System<sup>1</sup>

S. Olariu

J. Schwing, and J. Zhang

Department of Computer Science

Old Dominion University

Norfolk, VA 23529-0162

U.S.A.

**Abstract-** A bus system that can change dynamically to suit computational needs is referred to as reconfigurable. We present a fast adaptive convex hull algorithm on a 2-dimensional processor array with a reconfigurable bus system (2-d PARBS, for short). Specifically, we show that computing the convex hull of a planar set of  $n$  points taken  $O(\log n / \log m)$  time on a 2-d PARBS of size  $m \times n$  with  $3 \leq m \leq n$ . Our result implies that the convex hull of  $n$  points in the plane can be computed in  $O(1)$  time in a 2-d PARBS of size  $n^{1.5} \times n$ .

## 1 Introduction

Recent advances in VLSI have made it possible to build massively parallel machines featuring many thousands of cooperating processors. This increase in computational power does not, however, translate into increased performance of the same order of magnitude. One of the reasons seems to be that interprocessor communications and simultaneous memory accesses often act as bottlenecks in parallel machines.

To alleviate the inefficiency of long distance communication among processors, bus systems have been recently added to a number of parallel machines [2-4,5,6,11]. If such a bus system can be dynamically changed, under program control, to suit communication needs among processors, it is referred to as *reconfigurable*. Examples include the *bus automaton* [11], the *reconfigurable mesh*, and the *polymorphic torus* [2,3], among others.

The computational model used throughout this work is the *reconfigurable mesh* [5]. An  $m \times n$  reconfigurable mesh (also called a PARBS [13]) consists of  $m \times n$  identical processors positioned on a rectangular array (refer to Figure 1). The processor at  $(i, j)$ , ( $1 \leq i \leq m; 1 \leq j \leq n$ ) is identified by  $P(i, j)$ . Every processor has 4 ports denoted by  $N$ ,  $S$ ,  $E$ , and  $W$ . There are also implicit *north*, *south*, *east*, and *west* directions (refer to Figure 1). In each processor, ports can be dynamically connected in pairs to suit computational needs. In the absence of these local connections, the PARBS is functionally equivalent to the mesh connected computer.

<sup>1</sup>This work was supported by NASA under grant NCC1-99 by the National Science Foundation under grant CCR-8909996 is gratefully acknowledged

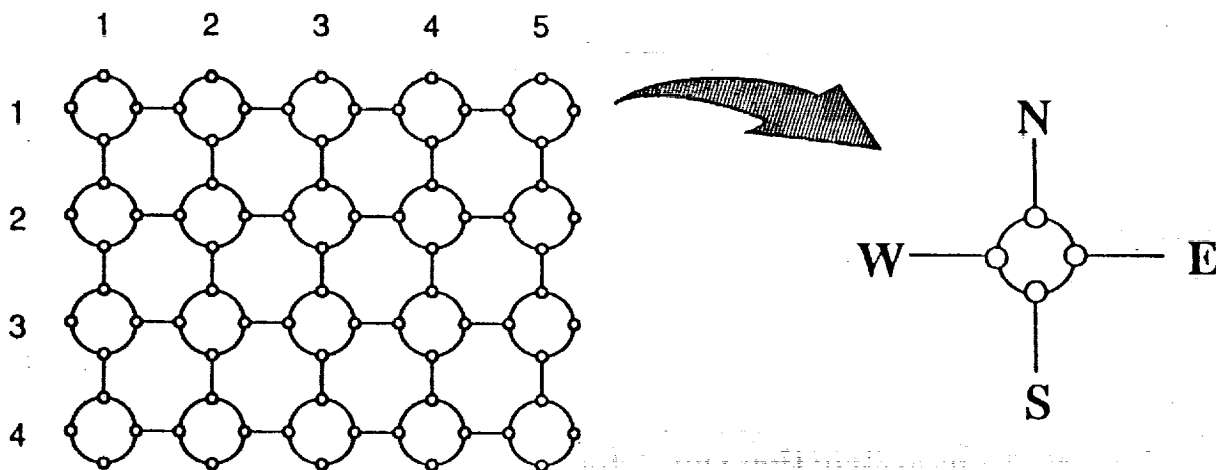


Figure 1: A 4x5 PARBS

We assume that each processor has a small number of registers of size  $O(\log n)$  bits and that a processor can perform in unit time standard arithmetic and boolean operations. We assume a single instruction stream: in each time unit the same instruction is broadcast to all processors, which execute it and wait for the next instruction. Each instruction can consist of setting local connections (as explained later), performing an arithmetic or boolean operation, broadcasting a value on a bus, or receiving a value from a specified bus. The regular structure of the PARBS makes it suitable for VLSI implementation. In fact, it has been argued [5] that the PARBS can be used as a universal chip capable of simulating any equivalent-area architecture without loss of time.

By adjusting the local connections within each processor several subbuses can be established. We assume that the setting of local connection is destructive in the sense that setting a new pattern of connections destroys the previous one. At any given time, only one processor can broadcast a value onto a bus. Processors, if instructed to do so, read the bus. If no value is being transmitted on the bus, the read operation has no result. It is assumed [5,6] that communications along buses take  $O(1)$  time. This seems to be a reasonable assumption in the light of recent experiments with the YUPPIE system [4].

A number of problems have been solved in  $O(1)$  time on PARBS. Very recently, Wang et al [13] have proposed  $O(1)$  algorithms for the transitive closure and some related graph problems; Olariu, Schwing, and Zhang [9] have proposed an adaptive sorting algorithm; specifically, they show that sorting a sequence of  $n$  reals takes  $O(\frac{\log n}{\log m})$  time on a 2-d PARBS of size  $nm \times n$  with  $3 \leq m \leq n$ . In particular, their result implies a constant-time sorting algorithm on an  $n^{1.5} \times n$  2-d PARBS.

The convex hull of a set of points in the plane is defined as the smallest area convex set that contains the original set. The problem of computing the convex hull of points in the plane is central in a variety of problems in pattern recognition, computer graphics, statistics, and image processing [1,7,8,10].



To the best of our knowledge, no convex hull algorithm has been reported in the literature on a 2-d PARBS. The purpose of this paper is to propose a fast adaptive convex hull algorithm for a set of  $n$  points in the plane. We reduce the problem of computing the convex hull of a set of planar points to the problems of sorting and computing the prefix maximum of  $n$  real numbers. To begin, we show that the problem of computing the maximum of  $n$  real numbers can be solved in time  $O(\frac{\log n}{\log m})$  on a 2-d PARBS of size  $m \times n$  with  $2 \leq m \leq n$ . We also use the fast adaptive sorting algorithm of [9]. What results is a fast adaptive algorithm that computes the convex hull of a set of  $n$  points in the plane in  $O(\frac{\log n}{\log m})$  time on a 2-d PARBS of size  $nm \times n$  with  $3 \leq m \leq n$ . In particular, for  $m=n^{0.5}$  we obtain an  $O(1)$  time convex hull algorithm on a 2-d PARBS of size  $n^{1.5} \times n$ .

## 2 The stepping stones

Our convex hull algorithm relies on a number of intermediate results that we present next. To begin, we consider the problem of computing the prefix maximum of  $n$  reals on a  $n \times n$  PARBS. Specifically, given  $n$  real numbers  $a_1, a_2, \dots, a_n$  with processor  $P(1, j)$  storing  $a_j$ , the problem is to compute  $\max_{1 \leq i \leq j} \{a_i\}$  for all  $1 \leq j \leq n$ . Our algorithm involves establishing a number of subbuses and broadcasting values along them. The details of our algorithm are spelled out by the following sequence of steps.

**Algorithm Prefix-Maximum;**

- Step 1.** every processor  $P(i, j)$  ( $2 \leq i \leq n-1; 1 \leq j \leq n$ ) connects its ports  $N$  and  $S$ ;
- Step 2.** every processor  $P(1, j)$  ( $1 \leq j \leq n$ ) broadcasts  $a_j$  southbound along the vertical subbus in column  $j$ ;
- Step 3.** every processor  $P(i, j)$  ( $2 \leq i \leq n-1; 2 \leq j < i$ ) connects its ports  $W$  and  $E$ ;
- Step 4.** every processor  $P(j, j)$  ( $2 \leq j \leq n$ ) broadcasts  $a_j$  westbound along the horizontal subbus in row  $j$ ;
- Step 5.** every processor  $P(j, i)$  ( $2 \leq j \leq n-1; 1 \leq i < j$ ) compares  $a_i$  and  $a_j$ ;  
     **if**  $a_i > a_j$  **then**  
          $P(j, i)$  disconnects the horizontal subbus;  
         marks itself;
- Step 6.** every marked processor  $P(i, j)$  broadcasts a "0" along the horizontal subbus eastbound;
- Step 7.** every processor  $P(j, j)$  ( $1 \leq j \leq n$ ) stores in its own memory a "0" or a "1" depending on whether or not it has received a "0" in Step 6;
- Step 8.** every processor  $P(i, j)$  ( $2 \leq i < j \leq n$ ) connects its ports  $N$  and  $S$ ;

**Step 9.** every processor  $P(j, j)$  ( $2 \leq j \leq n$ ) broadcasts on the vertical subbus northbound the value it has stored in Step 7;

**Step 10.** every processor  $P(1, j)$  ( $2 \leq j \leq n$ ) that has received a "0" in Step 9 connects its ports  $W$  and  $E$ ;

**Step 11.** every processor  $P(1, j)$  ( $1 \leq j \leq n$ ) that stores a "1" broadcasts  $a_j$  eastbound along the horizontal subbus in row 1;

**Theorem 1.** Algorithm Prefix-Maximum correctly computes the prefix maximum of  $n$  real numbers in  $O(1)$  time on an  $n \times n$  PARBS.

**Proof.** To begin, note that in Step 5, every processor  $P(i, j)$  ( $2 \leq i \leq n-1; 1 \leq j < i$ ) knows  $a_i$  and  $a_j$ . Further, it is easy to see that at the end of Step 7 processor  $P(j, j)$  ( $1 \leq j \leq n$ ) stores a "1" if, and only if,  $a_j$  is as least as large as  $a_i$  with  $i < j$ .

Consequently, every processor  $P(1, j)$  in row 1 that at the end of Step 9 stores a "0" knows that  $a_j$  cannot be the prefix maximum of  $a_i$  for  $i \leq j$ . In fact the prefix maximum of the first  $j$  real numbers  $a_1, a_2, \dots, a_j$  is stored by the first processor to the left of  $P(1, j)$  that stores a "1". The conclusion follows.  $\square$

Next, we show how to compute the maximum of  $n$  real numbers  $a_1, a_2, \dots, a_n$  on an  $m \times n$  PARBS with  $2 \leq m \leq n$ . Again, we assume that the numbers are stored one per processor such that for all  $j$  ( $1 \leq j \leq n$ ),  $P(1, j)$  stores  $a_j$ . The idea of our algorithm is to partition the original  $m \times n$  PARBS into subPARBS of size  $m \times m$ . To avoid tedious but inconsequential housekeeping details we assume that  $n$  is a power of  $m$ .

We partition the  $n$  columns into contiguous groups of  $m$  columns each and let the  $k$ -th subPARBS,  $M_k$ , ( $0 \leq k \leq n/m - 1$ ) consist of the columns  $km + 1, km + 2, \dots, km + m$ . As a preprocessing step, for all  $j$  ( $2 \leq j \leq n$ ) we move the data contained in  $P(1, j)$  to the "diagonal" processor of its  $m \times m$  subPARBS,  $P((j-1) \bmod m + 1, j)$ . The main loop of this algorithm applies the (prefix) maximum algorithm described above to specified  $m \times m$  subPARBS. This process proceeds iteratively, determining the maxima of groups of size  $m, m^2, m^3$ , and so on. Clearly, in  $\log_m n = \frac{\log n}{\log m}$  iteration we have computed the maximum of the  $n$  numbers.

We omit the details of bus-construction steps which are similar to those in the previous algorithm. The reader can easily fill in the details.

### Algorithm Maximum;

**Step 1.** {preprocessing}

**for all**  $j$  ( $1 \leq j \leq n$ ) **in parallel**

    establish a vertical subbus from  $P(1, j)$  to  $P((j-1) \bmod m + 1, j)$ ;

$P(1, j)$  broadcasts  $a_j$  on this subbus to  $P((j-1) \bmod m + 1, j)$ ;

$P((j-1) \bmod m + 1, j)$  marks itself

**endfor**;

**Step 2.** {main loop}

  for  $k \leftarrow 1$  to  $\frac{\log n}{\log m}$  do

    for all  $j$  ( $1 \leq j \leq \frac{n}{m^k}$ ) in parallel

      all processors connect ports  $W$  and  $E$ ;

      all processors  $P(i, (j-1)m^k + 1)$  split the horizontal subbus in row  $i$ ;

      all marked processors broadcast the value they hold

      along the horizontal subbus westbound;

      all marked processors unmark themselves;

$M_{(j-1)m^{k-1}}$  computes the maximum of the values

      in column  $(j-1)m^k + 1$ ;

      let the result be stored in  $P((j-1) \bmod m + 1, (j-1)m^k + 1)$ ;

      all processors  $P((j-1) \bmod m + 1, (j-1)m^k + 1)$  mark themselves

    endfor

  endfor;

**Theorem 2.** Algorithm Maximum correctly computes the maximum of  $n$  real numbers in  $O(\frac{\log n}{\log m})$  time on an  $m \times n$  PARBS with  $2 \leq m \leq n$ .

**Proof.** The correctness is implied by the following result: at the end of the  $t$ -th iteration ( $0 \leq t \leq \frac{\log n}{\log m}$ ), for all  $j$  ( $1 \leq j \leq \frac{n}{m^t}$ ), processor  $P((j-1) \bmod m + 1, (j-1)m^t + 1)$  contains the maximum in columns  $(j-1)m^t + 1$  through  $jm^t$ .

The proof of the above statement is by induction on  $t$ . The basis is easy: at the end of the 0-th iteration the conclusion is guaranteed by the preprocessing step.

Assume the above statement satisfied at the end of the  $t$ -th iteration. We only need show that it also holds at the end of the  $(t+1)$ -st iteration. For this purpose, it is instructive to follow the  $(t+1)$ -st iteration: here, after all processors connect their ports  $W$  and  $E$  thus establishing horizontal subbuses in each row, the processors  $P(i, (j-1)m^{t+1} + 1)$  split the horizontal subbus in row  $i$ ; next, all marked processors broadcast the value they hold along the horizontal subbus westbound. By the induction hypothesis, these are processors  $P((j-1) \bmod m + 1, (j-1)m^t + 1)$ . Therefore, when the subPARBS  $M_{(j-1)m^{t+1}}$  compute the maximum of the values in column  $(j-1)m^{t+1} + 1$ , the induction hypothesis guarantees that the resulting value is the maximum in columns  $(j-1)m^{t+1} + 1$  through  $jm^{t+1}$ , a total of  $m^{t+1}$  columns.

To argue for the running time, note that by Theorem 1, the inner for loop runs in  $O(1)$  time. The conclusion follows.  $\square$

### 3 The Algorithm

We are now in a position to present our planar convex hull algorithm. Let  $S = \{p_1, p_2, \dots, p_n\}$  be a planar set of points; for  $1 \leq i \leq n$ ,  $p_i$  is represented by its Cartesian coordinates  $(x_i, y_i)$ . To avoid tedious details we assume, without loss of generality, that the points in  $S$  are in *general position*, with no three collinear and no two having the same  $x$  or  $y$  coordinate. The output of the convex hull algorithm is a linked list  $CH$  that contains all the points

on the convex hull starting with the one with the largest  $x$  coordinate and proceeding counterclockwise. Our algorithm consists of the following sequence of steps.

**Algorithm Convex-Hull;**

**Step 1.** find the four extremal points in  $S$ , and let them be, without loss of generality,  $p_1$ ,  $p_2$ ,  $p_3$ , and  $p_4$ . Specifically,  $x_1 = \max_{1 \leq j \leq n} \{x_j\}$ ,  $y_2 = \max_{1 \leq j \leq n} \{y_j\}$ ,  $x_3 = \min_{1 \leq j \leq n} \{x_j\}$ , and  $y_4 = \min_{1 \leq j \leq n} \{y_j\}$ .

**Step 2.** compute the sets

$$\begin{aligned} S_1 &= \{p_i | x_2 \leq x_i \leq x_1; y_1 \leq y_i \leq y_2\}, \\ S_2 &= \{p_i | x_3 \leq x_i \leq x_2; y_3 \leq y_i \leq y_2\}, \\ S_3 &= \{p_i | x_3 \leq x_i \leq x_4; y_4 \leq y_i \leq y_3\}, \\ S_4 &= \{p_i | x_4 \leq x_i \leq x_1; y_4 \leq y_i \leq y_1\}. \end{aligned}$$

**Note:** For simplicity, we deal with  $S_1$  only, the others being perfectly similar.

**Step 3.** sort the points in  $S_1$  by increasing  $y$  coordinate, and let  $L_1 = (p_1 = q_1, q_2, \dots, q_t = p_2)$  be the resulting sorted sequence;

**Step 4.** for all  $j$  ( $1 \leq j \leq t$ ) in parallel

find the subscript  $d_j$  ( $j < d_j \leq t$ ) such that the angle determined by  $q_{d_j}$ ,  $q_j$ , and the negative direction of the  $x$  axis is as large as possible;

**Step 5.** compute the prefix maximum of the values  $d_j$  in  $L_1$ , and set  $m(j) \leftarrow \max_{1 \leq i \leq j-1} \{d_i\}$ ;

**Step 6.**  $CH_1 \leftarrow L_1$ ;

for all  $j$  ( $2 \leq j \leq t-1$ ) in parallel

remove  $q_j$  from  $CH_1$  whenever  $d_j \leq m(j)$ ;

Before giving the proof of correctness of our algorithm, we need to take note of the following simple observation. The sorted sequence  $L_1$  of points obtained at the end of Step 3 can be viewed as determining a *polygonal line* (termed a *chain* in [10]) joining  $p_1$  and  $p_2$ . It is easy to see that the convex hull  $CH$  of the set  $S$  of points is exactly the convex hull of the simple polygon  $P$  obtained by concatenating the polygonal lines  $L_1$ ,  $L_2$ ,  $L_3$ , and  $L_4$ , in this order.

The following result argues for the correctness of our algorithm.

**Theorem 3.** At the end of Step 6,  $CH_1$  contains the portion of the convex hull contained in  $S_1$ .

**Proof.** By the previous observation we only need show that the linked list  $CH_1$  obtained at the end of step 6 contains the restriction of the convex hull of  $P$  between  $p_1$  and  $p_2$ . This follows from the following claim

a point  $q_j$  ( $2 \leq j \leq t-1$ ) of  $L_1$  belongs to  $CH$  if, and only if,  $d_j > m(j)$ .

First, let  $q_j$  ( $2 \leq j \leq t-1$ ) in  $L_1$  belongs to the convex hull and let  $q_i$  and  $q_k$  ( $i < j < k$ ) be its immediate neighbors on the convex hull. (We note that since  $q_1$  and  $q_t$  trivially belong to the convex hull, the points  $q_i$  and  $q_k$  are well defined.) Clearly,  $d_i=j$  and so  $m(j) = j < d_j = k$ , as claimed.

Conversely, if some point  $q_j$  in  $L_1$  does not belong to the convex hull then let  $q_i$  and  $q_k$  ( $i < k$ ) be the closest points on the convex hull, with  $q_j$  lying on the chain from  $q_i$  to  $q_k$ . Since  $q_i$  and  $q_k$  are neighbors on the convex hull, we have  $d_i=k$ ; furthermore,  $d_j \leq k = m(j)$ , and the conclusion follows.  $\square$

Next, we propose to show how Steps 1-6 above can be efficiently implemented on a 2-d PARBS. More precisely, we assume a 2-d PARBS of size  $nm \times n$  with  $3 \leq m \leq n$ . Some of the Steps 1-6 in our algorithms need the whole PARBS while others can run on a subPARBS, as specified; the data movement necessary to conform to the input requirements of a specific step are ignored here; the reader can easily work out all the details.

Step 1 can be implemented to run in  $O(1)$  time on an  $n \times n$  subPARBS since we only need compute  $\max_{1 \leq j \leq n} \{z_j\}$  and  $\min_{1 \leq j \leq n} \{z_j\}$  with  $z = x$  and  $z = y$ .

Step 2 is demonstrated for  $S_1$  only; computing  $S_i$  with  $i = 2, 3, 4$  is similar. All that is needed is to establish a subbus running through the whole of row 1. The processors storing  $p_1$  and  $p_2$  broadcast, in two computational steps, their Cartesian coordinates to all processors in row 1; every processor that stores a point in  $S_1$  marks itself. Thus Step 2 runs in  $O(1)$  time.

Step 3 can be implemented as follows. First, all unmarked processors change the  $y$  coordinate of the point that they store to  $+\infty$ . Now the sorting algorithm in [9] is invoked: this runs in  $O(\frac{\log n}{\log m})$  and uses the whole PARBS. Note that at the end of Step 3, processors  $P(1,1), P(1,2), \dots, P(1,t)$  contain  $L$  in sorted order.

Step 4 can be implemented to run in  $O(1)$  time on an  $mn \times n$  subPARBS as follows. Recall from Step 3 that, initially, for all  $1 \leq j \leq t$   $P(1,j)$  stores  $q_j$ . For further reference, this subPARBS is further subdivided into subPARBS of size  $m \times n$  as follows. The first  $m \times n$  subPARBS involves the first  $m$  rows, the second the next  $m$  rows and so on. We establish vertical subbuses in each column and let  $P(1,j)$  broadcast the Cartesian coordinates of  $q_j$  along the subbus in column  $j$  ( $1 \leq j \leq t$ ). Next, establish horizontal subbuses running from  $P(m(j-1)+1,j)$  to  $P(m(j-1)+1,t)$  ( $1 \leq j \leq t$ ). Note that these are precisely the first rows of our  $m \times n$  subPARBS. For all  $j$ ,  $P(m(j-1)+1,j)$  broadcasts the Cartesian coordinates of  $q_j$  eastbound on the horizontal subbus in row  $m(j-1)+1$ . Every processor  $P(m(j-1)+1,k)$  with  $j < k \leq t$  computes the angle specified in Step 4. Actually, computing the angle itself is not necessary, the tangent of the angle can be readily computing using two subtractions and a division. Now the maximum of all values in the first rows of these subPARBS can be computed in  $O(\frac{\log n}{\log m})$  time using Algorithm Maximum developed in Section 2. It is easy to arrange for the maximum in row  $m(j-1)+1$  to be sent back to  $P(1,j)$ . This, clearly takes  $O(1)$  time since only the appropriate subbuses have to be established and the information broadcast along them.

Step 5 can be implemented to run on an  $n \times n$  subPARBS by using Algorithm Prefix-Maximum discussed in Section 2.

Step 6 involves marking every  $P(1, j)$  that contains a point of the convex hull. After this is done, a horizontal subbus is established in row 1. Every marked processor splits this bus and broadcasts its identity westbound on its own subbus. This, in fact creates the list  $CH_1$  as desired. Clearly, the running time of this step is  $O(1)$ .

To summarize our discussion we state the following result.

**Theorem 4.** The convex hull of a planar set of  $n$  points can be computed on an PARBS of size  $nm \times n$  with  $3 \leq m \leq n$  in  $O(\frac{\log n}{\log m})$  time.  $\square$

In particular, if  $m = n^{0.5}$  then we have the following result.

**Corollary 4.1.** The convex hull of a planar set of  $n$  points can be computed in  $O(1)$  time on an PARBS of size  $n^{1.5} \times n$ .  $\square$

## 4 Conclusion

A bus system that can be dynamically altered to suit communicational needs among co-operating processors is referred to as *reconfigurable*. In this paper we a fast adaptive algorithm to solve the planar convex hull problem.

Specifically, we showed that computing the convex hull of a set of  $n$  points in the plane takes  $O(\frac{\log n}{\log m})$  on a 2-d PARBS of size  $nm \times n$  with  $3 \leq m \leq n$ . In particular, our result implies that the same problem can be solved in  $O(1)$  time on a 2-d PARBS of size  $n^{1.5} \times n$ .

## References

- [1] J. A. Holey and O. H. Ibarra, Iterative algorithms for planar convex hull on mesh connected arrays, *Proc. 1990 International Conference on Parallel Processing*, III-102-III-109.
- [2] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Transactions on Computers*, vol. C-38, no. 9, (1989) 1345-1351.
- [3] H. Li and M. Maresca, Polymorphic-torus architecture for computer vision, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 3, (1989) 233-243.
- [4] M. Maresca and H. Li, Connection autonomy and SIMD computers: a VLSI implementation, *Journal of Parallel and Distributed Computing*, vol. 7 (1989) 302-320.
- [5] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, Meshes with reconfigurable buses, *Proceedings of the International Conference on Parallel Processing*, vol. 1, (1988) 205-208.

- [6] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, Data movement operations and applications on Reconfigurable VLSI arrays, *Proceedings of the fifth MIT Conference on Advanced Research in VLSI*, (1988) 163-178.
- [7] R. Miller and Q. Stout, Mesh Computer Algorithms for Computational Geometry, *IEEE Trans. on Computers*, 38 (1989), 321-340.
- [8] R. Miller and Q. Stout, Efficient Parallel Convex Hull Algorithms, *IEEE Trans. on Computers*, 37 (1988), 1605-1618.
- [9] S. Olariu, J. L. Schwing, and J. Zhang, A Fast Adaptive Sorting Algorithm on a Two Dimensional Processor Array with a Reconfigurable Bus System, submitted.
- [10] F. P. Preparata and M. I. Shamos, Computational Geometry, An Introduction, Springer-Verlag, New York, Berlin, 1988.
- [11] J. Rothstein, Bus automata, brains, and mental models, *IEEE Trans. on Systems Man Cybernetics*, 18, (1988).
- [12] Q. F. Stout, Meshes with Multiple Buses, *Proc. 27th IEEE Symp. on the Foundations of Comp. Science*, (1986) 264-273.
- [13] B. F. Wang, C. J. Lu, and G. H. Chen, Constant Time Algorithms for the Transitive Closure Problem and its Applications, *Proceedings of the International Conference on Parallel Processing*, vol. 3, (1990) 52-59.

10. 11. 2001

[illegible]

..

•

•

2

—

—



# VHDL Simulation with Access to Transistor Models

J. Gibson

NASA SERC for VLSI Systems Design

University of Idaho

Moscow, Idaho 83843

## 1 Introduction

Hardware description languages such as VHDL have evolved to aid in the design of systems with large numbers of elements and a wide range of electronic and logical abstractions. For high performance circuits, behavioral models may not be able to efficiently include enough detail to give designers confidence in a simulation's accuracy. One option is to provide a link between the VHDL environment and a transistor level simulation environment. The coupling of the Vantage Analysis Systems VHDL simulator and the NOVA simulator provides the combination of VHDL modeling and transistor modeling.

## 2 Vantage VHDL Simulator

The Vantage Analysis Systems VHDL simulation environment is a full implementation of the IEEE 1076 VHDL Standard. The Vantage system is entirely written in "C". Hierarchical designs from Mentor Graphics' NetEd, EDIF or other schematics can be imported into Vantage. The Vantage system compiles VHDL designs and simulates their behavior either interactively or in a batch mode. Incremental symbol or schematic changes can be made in the Vantage environment, from which structural VHDL can be automatically be generated. The created or edited schematics can be exported back into the original schematic environment. Connectivity and structural checks are made by the schematic viewer.

Results may be viewed as waveforms or as entries in a table, and can be viewed as the simulator is running or after the simulation run has completed. Circuit node values can also be displayed on the associated node in the circuit schematic.

The Vantage simulation control gives the user source code level breakpoints and triggering capability based on a very wide range of conditions specified by the user. Convenient viewing and the manipulation of the values of signals, variables and constants is provided. Breakpoints can be based on boolean expressions, change in a signal, source code lines, or by design units (by instance or globally).

In the Vantage system, the VHDL source code is parsed by a C code generator. Then the host "C" code compiler prepares the executable file or files. The Vantage Intermediate Format is used for the generation of the "C" code. All design units lower in hierarchy must have older time stamps than the designs that reference them. The Vantage system automatically recompiles all "out-of-date" design units referenced during a recompile.

Several conveniences are provided by the Vantage system, including automatic mapping

of signal names that do not conform to the VHDL signal name convention to an internal, VHDL compatible form. This permits familiar names of existing systems to be used when interfacing with the Vantage simulator. The Vantage Control Language facilitates the generation of test vectors.

Extensive libraries of vendor supplied VHDL models, parts and packages, from SSI to VLSI, are available. Any VHDL in a Vantage library can be exported to an ascii file. Vantage also supplies a concurrent compiler that spreads the compilation task of a design across a network, to improve compilation speed.

### 3 The NOVA Simulator

NOVA is a logic simulator that was recently developed at the University of Idaho NASA Space Engineering Research Center for VLSI Systems Design. NOVA is a second generation design, targeted for designs of up to a few million primitives (transistors and logic gates). NOVA has been used to simulate integrated circuits designed for the NASA Space Station and Explorer missions and other NASA projects, and for Hewlett Packard disk and tape drives. NOVA presently is ported to HP 9000 Series 300, 400 and 700's, the HP/Apollo DN10000, the Cray X-MP and NeXT systems. Behavioral models can be used in NOVA to assist in the architectural definition of major functional blocks before circuit details are completely known, and to improve simulation performance at most levels of system modeling.

Structural description is accomplished using the BOLT or HP Block Description (BDL) languages. NOVA utilizes hierarchical design methodology, allowing designs to be conveniently partitioned. Efficient management of design complexity is made possible by the block oriented circuit description. SCIP schematics, based on Hewlett Packard design tools, provide schematic documentation, from which the BOLT design description can be extracted.

NOVA supports synchronous and asynchronous modeling of hierarchical designs using logic primitives and intrinsic devices. Most types of transistors and logic gates are represented in the existing library, including bidirectional CMOS devices and bidirectional transmission gates. If the designer requires new primitives to accurately model a special circuit, NOVA provides a means of incorporating the user defined model.

NOVA provides for full timing analysis of combinational and sequential circuits, specified by rise and fall delays, using a timing wheel based simulation engine.

Behavioral modeling, using a "C" based functional model capability, allows the designer to generate high level descriptions of a block of circuitry. Productivity is improved by allowing the designer to simulate the function of a block before the detailed circuit implementation is available. Very good behavioral modeling performance is achieved by compiling the functional models with the simulation engine. Transistor or logic gate circuit models can be mixed with, or replace, behavioral models, with full timing and delay modeling capability.

A configurable X11 graphics user interface assists the designer in viewing and interpret-

ing simulation results. Signals and busses can be viewed as waveforms. Trigger conditions can be defined to find specific signal relationships in the simulation output, allowing in depth analysis of complex events.

NOVA also provides numerous analysis features, such as node coverage, simulation debugging, output formatting, node forcing, simulation state logging and saving, and others.

Software tools in NOVA greatly simplify test vector development for design verification. A test vector programming language makes it possible for the designer to develop compact descriptions of complex simulation sequences.

The overall capability of the NOVA simulator closely matches that of many commercial simulators. A major issue is the fact that NOVA's behavioral modeling capability is not close to any industry standards, like VHDL, which makes it difficult for NOVA users to leverage existing model libraries or to import existing designs. On the other hand, the transistor level modeling in NOVA is highly evolved and well known by the NOVA user community. Using VHDL as a modeling language will likely open many opportunities for an organization like the NASA SERC for VLSI Systems Design, compared to using a proprietary modeling language.

## 4 Multiple Value Logic Systems

Simulations performed at the logic level of abstraction describe a digital circuit in terms of primitive logic functions such as NAND, NOR, etc., and allow for the nets interconnecting the logic functions to carry states of zero, one, unknown, and high impedance. In the case of NOVA, strengths of active, resistive and floating accompany the logic states, to provide a total of twenty-two logic values. These twenty-two logic values are built into the structure of the simulator and are possible to change or expand, but not necessarily easily.

The VHDL standard provides a Multi-Value Logic structure that allows the individual user of VHDL to tailor the resolution of logic values to satisfy the needs of a general design methodology or the specific preferences of individuals. This flexibility in describing logic values in a digital system can have a considerable influence on the transportability of a VHDL model from one design group to another. Having too few logic values can cause erroneous results in hardware systems that have bidirectional data busses, open-collector or high-impedance conditions. More logic values are necessary to model open-collector devices with pull-up resistors and situations that can occur when initializing a digital system.

For the coupling of NOVA and the Vantage VHDL simulator, the model compatibility issue is resolved by using the same types of logic states and strengths for both simulation systems. The Multi-Value logic system of the Vantage system is set to represent the same values as NOVA, with the resolution functions providing the same logic value when circuit outputs are connected together.

## 5 Transistor Level Performance Issues

One of the primary motivations for this work is the acceleration of the performance of transistor, or switch level, simulations in the VHDL environment. During the original design of the VHDL language, transistor level simulation was not included as a primary requirement. However, algorithms have been developed in VHDL that can simulate the properties of bidirectional transmission gates, without extensions to the VHDL language.

Given that VHDL can model bidirectional pass transistor networks, a second issue is the amount of memory required to represent a primitive in a VHDL simulator compared to a more "hardwired" simulator that has semantics built into it's runtime kernel, such as NOVA or Verilog. NOVA uses about 35 bytes per primitive (transistor) in the internal representation. The amount of memory required for the average primitive in VHDL is not as easily determined. Based on overall file size, it appears that as much as 1000 bytes of data are associated with each primitive in the Vantage simulation system, a VHDL system that is known for relatively good performance. It is common for VHDL system models to require virtual memory, which automatically invokes at least a 10X performance penalty, relative to a simulation model that runs entirely in RAM. A 500,000 NOVA transistor model, entirely composed of transistor primitives without any use of behavioral models, will fit in a workstation's 32 megabyte random access memory.

Another difference between the Vantage VHDL modeling system and NOVA is the use of resolution functions. A resolution function, applied to a node in a circuit, is used to return the value of a signal when the signal is driven by multiple drivers, during a simulation. All VHDL signals with multiple drivers must have a resolution function tied to that signal. With VHDL, the designer has the capability of defining any type of resolution function desired, either wired-OR, wired-AND or average signal value. NOVA has the equivalent of a resolution function, but it is coded in optimized "C", tightly linked with the rest of the core of the simulator and is fixed in definition. A VHDL resolution function is written in VHDL as a package, a representation that will be translated into "C" code but not optimized for fast execution.

## 6 "C" Behavioral Model Interface to the Vantage Simulator

A complete simulation system should be able to efficiently and quickly incorporate algorithms not represented in the native language of the simulator. To make possible fast development of designs, algorithms that already exist in program form should be usable in system simulations without requiring a reimplementaion. If a design under development uses, for example, output from a digital filtering algorithm that is going to be part of another integrated circuit, it may be of considerable advantage to use a high level programming language version of that algorithm. Developing a new implementation of a digital filtering algorithm is not only a duplication of effort; new sources of error and changes in performance may also result.

Both NOVA and the Vantage Simulator allow "C" based behavioral models to be compiled into each simulation environment. In NOVA, a BOLT description is written for the behavioral model, describing the input and output connections to the rest of the simulation model. The NOVA "C" behavioral model is compiled in "C" and the resulting object data is compiled with NOVA into a form containing the regular primitive based simulation environment and the functional model. In the Vantage VHDL environment, the process is similar, in that the user must provide an entity (the structural or input/output description) written and compiled in VHDL prior to compiling the combined VHDL/C architecture. The architecture (the behavior of the module) is written and compiled in "C". Both systems provide the necessary parameters required for passing state, strength and timing information between the behavioral model and the main simulation model. Again, using an industry standard language, such as VHDL, as the modeling interface, should provide more flexibility and opportunity in the future.

## **7 "C" Based Simulator Interface to the Vantage Simulator**

Using the Vantage Simulator and NOVA is one way of meeting the dual goals of using an industry standard behavioral modeling language and achieving decent transistor level simulation performance. It is presently estimated that the transistor level simulation performance of NOVA will exceed that of the Vantage Simulator by 20 to 50 times. The software to accomplish the link between NOVA and the Vantage Simulator is expected to be available from Vantage in the near future.

## **8 Future Directions**

Research is in progress to identify simpler behavioral modeling methodologies that are quicker and easier to use. The objective is to reduce design time by having only one complete representation of a design, first as a top level behavioral model which is then broken in to subsystems of the design as the function of each block is identified. At any time, either the behavioral or transistor level representation of a block can be used. For performance reasons, the behavioral models can be used. For detailed circuit timing and performance analysis, the transistor level representations of the blocks being designed can be used while the rest are left at the behavioral level. VHDL is satisfactory as the modeling language for this effort, since description standards will be a large part of the solution to an easier to use simulation environment. In a complementary effort, software tools are being developed by the University of Idaho Computer Science Department to compare the functionality of behavioral models and transistor level models.

